UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN ESPECIALIDAD EN SONIDO E IMAGEN

PROYECTO FIN DE CARRERA

RECONOCIMIENTO E IDENTIFICACIÓN DE LOGOTIPOS EN IMÁGENES CON TRANSFORMADA SIFT

Autor: CÉSAR JUÁREZ MEGÍAS

Tutor: JULIO VILLENA ROMÁN
(Departamento de INGENIERÍA TELEMÁTICA)

Leganés, Diciembre de 2011

Título: RECONOCIMIENTO E IDENTIFICACIÓN DE LOGOTIPOS EN IMÁGENES CON TRANSFORMADA SIFT

Autor: CÉSAR JUÁREZ MEGÍAS

Director: JULIO VILLENA ROMÁN

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 2011 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

AGRADECIMIENTOS

No solo con el estudio se consigue lo que se quiere, siempre se necesita un apoyo de las personas que te rodean, consejos, ánimos, etc, es por esto por lo que necesito que quede constancia del agradecimiento que siento ante esa ayuda desinteresada.

Gracias a mis padres por soportarme económica y moralmente durante estos largos años.
Gracias también mis hermanos por estar ahí siempre y por ser como son.
A mis abuelos por ser mis segundos padres.
A mis familiares, aunque no os nombre a todos estáis en mi cabeza: Ana, Adrián, Javi, Manu, Andrés, Almudena, Rocío, Macu, Yoli, Iván, Carmen, Carlos, etc.
A mi tutor, Julio, por apoyarme y darme la oportunidad de desarrollar este sistema tan interesante.

Gracias a mi familia política, por acogerme en su casa y por estar tan pendiente de mí en todo momento (Maribel, Domingo, David, Alberto, Miriam, Ángela, Miguel, Rubén, Javier, María José, etc)

A mis amigos de la carrera, compañeros de prácticas, de clases, de laboratorios, etc.
Gracias por ayudarme a seguir y no tirar la toalla y gracias porque no os considero compañeros, os considero amigos: Jessi, Pablo, Efrén, Estela, David, Carlos, Dani, Cristina, Irene, Eli, etc.
Seguro que me dejo alguno, pero aunque no los nombre ellos saben que los tengo en mente.

A mis compañeros de la beca del Espacio Estudiantes, menudo equipazo teníamos: Jaime, Marco, Daniel, Pepe, etc.
A mis amigos (del barrio, colegio, instituto), porque también habéis estado ahí cuando he necesitado salir un rato para despejarme: Alberto, Santiago, Dani, Carlos, Marta, Bea, Sergio, Lalo, Alberto, Ángel, etc.

A los que ya no están, pero que siempre estarán…

A los que se me olvide en este momento mencionar, pero que en algún momento me hayan ayudado a seguir mi camino sin mirar atrás, a tomarme la vida de otra manera, a ser yo.

Por último pero no menos importante, gracias a Lidia por apoyarme en todo momento, por regalarme su tiempo, por ayudarme a ser mejor persona y sobre todo por aguantarme.

RESUMEN

Mediante la Transformada SIFT, se pueden definir los puntos clave que caracterizan a una imagen cualquiera que luego podrán ser localizados en otras escenas en las que existen rotaciones, cambios de escala e iluminación y oclusiones parciales. De esta forma se podrá realizar una búsqueda automática de objetos en distintas imágenes.

Durante el desarrollo de este trabajo se realizará una investigación de los métodos que la Transformada SIFT de David Lowe permite, y se buscarán una serie de conclusiones utilizando dichos métodos en aplicaciones destinadas a la identificación de logotipos corporativos. Para ello se realizará la implementación de un pequeño sistema que será capaz de entrenarse y realizar una comprobación de su funcionamiento. Son muchas las implementaciones de este algoritmo, realizadas para los distintos lenguajes de programación. En este caso se utilizará la versión para Matlab denominada VL_FEAT.

En un primer lugar se realiza un entrenamiento, utilizando una serie de imágenes que contendrán los logotipos y que estarán almacenadas en un directorio (al menos 100). Mediante la extracción de los puntos clave de dichas imágenes con el método de Lowe, se obtienen una serie de características que se almacenarán en una base de datos para su posterior uso en el reconocimiento de estos logotipos.

Una vez realizado el almacenamiento de las imágenes, se utilizará de nuevo otro de los métodos del paquete de herramientas VL_FEAT para encontrar las coincidencias entre la imagen con la que queremos comprobar el funcionamiento del sistema y las almacenadas anteriormente en el apartado de entrenamiento. Finalmente tras diversas comprobaciones, se realizan una serie de conclusiones que definirán el uso de los mejores métodos de esta Transformada para el uso aquí expuesto.

Palabras clave: reconocimiento de imágenes, SIFT, David Lowe, VL_FEAT…

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1.- INTRODUCCIÓN Y OBJETIVOS

1.1.- INTRODUCCIÓN

El mundo de las aplicaciones que utilizan actualmente el tratamiento de imágenes es bastante extenso, y por tanto es muy importante su estudio para un futuro desarrollo de herramientas.

Hoy en día, debido a la gran cantidad de información multimedia creada y modificada diariamente, no sería posible realizar una búsqueda manual de objetos que aparezcan en dichos archivos, o más aún del número de veces que esos objetos se repiten. Tales sumas son imposibles de tratar manualmente, por lo que estas tareas se deberían realizar mecánicamente.
Para que se comprenda tal cantidad de información, he aquí dos claros ejemplos:

- 2 millones de vídeos se agregan diariamente a YouTube (el promedio son vídeos cuya duración es menor a tres minutos).
- 5 millones de imágenes se agregan diariamente a Flickr, es decir 60 fotos por segundo.

En un primer momento, se utiliza un etiquetado de las imágenes donde se comenta de qué trata la ilustración, lo que en ella aparece, etc. Al igual que es muy fácil explicar sobre qué trata una imagen, también es muy sencillo añadir comentarios sobre contenidos que no aparecen, es decir, falsear ese etiquetado. Es por esto, por lo que el reconocimiento de objetos en imágenes es tan demandado a día de hoy y desde hace unos años.

Google ha desarrollado una herramienta que (entre otras cosas) reconoce lugares, etiquetas de botellas de vino, logotipos, etc. denominada Google Goggles. ¿Por qué no seguir por ese camino e investigar un poco ese tipo de reconocimiento a veces tan útil?

Existe un algoritmo desarrollado en 1999 por David Lowe, que permite de obtener una serie de características invariables a la escala y a la rotación de una imagen dada. Es la denominada Transformada SIFT (Scale-Invariant Feature Transform). Entre las aplicaciones de esta herramienta se pueden destacar el reconocimiento de objetos, la navegación mediante mapas, reconocimiento de gestos, modelado en 3D, seguimiento de vídeos, detección de movimientos, etc. La cantidad de paquetes de herramientas y la extensión de dicha Transformada a los distintos lenguajes de programación existentes, hacen que este instrumento sea perfecto para el reconocimiento de objetos.

Por ejemplo, este trabajo tiene que ver con la proximidad de las elecciones generales en España, pues todos los principales partidos políticos tienen el derecho de obtener la misma cantidad de tiempo publicitario, número de anuncios en periódicos, etc. Hasta hoy, el “conteo” del número de páginas dedicadas a cada partido se hace manualmente; pero, ¿no sería posible encontrar mecánicamente, el logotipo de cada partido en una página de periódico?

Estas razones aquí expuestas crearon un entorno ideal de investigación, y por tanto la idea terminó gestándose como un reconocimiento de un conjunto de logotipos mediante las técnicas expuestas en la Transformada SIFT.

1.2.- OBJETIVOS Y MEDIOS EMPLEADOS

Los objetivos principales de este trabajo se basan en la creación e investigación de un sistema de reconocimiento de logotipos en imágenes capaz de almacenar las características y posteriormente localizarlas en entornos de desorden, distinta iluminación, etc., basándose en las aplicaciones existentes de la Transformada SIFT para múltiples imágenes.

A continuación se expondrán una serie de conclusiones del uso de los distintos métodos de extracción y localización de características, para alcanzar una solución óptima en el entorno del reconocimiento de logotipos.

Tras realizar una investigación sobre los paquetes de herramientas de la Transformada SIFT disponibles hoy en día en el mercado, se elige desarrollar el prototipo en Matlab por el previo conocimiento del mismo, la rapidez que muestra ante operaciones matemáticas con matrices y el paquete de herramientas asociado a este lenguaje de programación. El "*toolbox" (paquete de herramientas)* asociado al programa recibe el nombre de VL_FEAT.

Este paquete de herramientas tiene una gran versatilidad de uso para las distintas aplicaciones que se podrían intentar desarrollar y una gran capacidad de explotación del cómputo del programa. Entre los sistemas desarrollados con esta herramienta se encuentra el desarrollado por los autores de VL_FEAT, en el que se reconocen imágenes entre 101 clases posibles. Para probar dicho sistema se usó un 15% imágenes de entrenamiento de cada una de las categorías y se obtuvieron unos niveles de 65% de aciertos.

Imagen 1 – Grupos de imágenes utilizados en la herramienta desarrollada por VL_FEAT.

1.3.- FASES DE DESARROLLO

La primera fase del desarrollo es la búsqueda de información sobre la Transformada SIFT. Dentro de esta fase, el comprender cómo funciona SIFT, cómo realiza una búsqueda de las características más importantes de las imágenes, las almacena, las describe, etc, es esencial para el posterior desarrollo de la aplicación.

Tras una búsqueda exhaustiva de información, aparece el momento de la elección del paquete de herramientas, y a su vez del lenguaje de programación que se usará durante la implementación del sistema. En el caso de este proyecto, tal y como ya se ha comentado en el apartado de "objetivos y medios empleados", se realiza la elección del paquete de herramientas VL_FEAT para Matlab.

El siguiente punto a destacar es la investigación de los métodos y algoritmos contenidos en VL_FEAT y su comprensión. Para ello, son muchas las pruebas y las indagaciones en la ventana de comandos de Matlab, pues no siempre con la ayuda del programa es suficiente. Esta parte del proceso lleva un tiempo que no se reconoce del todo en el proceso de creación del programa, pero que es necesario y cabe destacar.

Posteriormente, es necesario realizar un estudio sobre cómo se van a desarrollar las distintos elementos del programa. Se observan qué módulos se utilizarán varias veces para crear un archivo propio de esa función, cuántas ventanas se deberán mostrar en pantalla y qué actividades se podrán realizar en cada una de ellas.

Tras de la creación de un esquema de implementación, se debe ir creando cada uno de los archivos que serán necesarios para el programa. Se rellenan cada uno de estos ficheros y finalmente, se crea la interfaz gráfica.

Revisar el contenido y la funcionalidad de cada uno de los archivos por separado es vital para la siguiente fase del proyecto que es la de proporcionar unos vínculos entre cada archivo para su funcionamiento global. Esta parte del proceso es muy útil pues se observan las comunicaciones de unos algoritmos con otros y es posible realizar los cambios pertinentes para que todo funcione de forma correcta.

La última fase de realización del programa es la comprobación y las pruebas a las que se someterá el sistema. Aquí se podrán ajustar los parámetros y umbrales necesarios para obtener unos buenos resultados y será posible observar si falta alguna utilidad, y es necesario añadirla al reconocedor.

La creación de una buena memoria es la última parte del proyecto, pues se debe redactar por qué se ha elegido el tema sobre el que se ha investigado, cómo se ha llevado a cabo, analizar de forma objetiva el programa y comentar de forma exhaustiva los resultados obtenidos.

1.4.- ESTRUCTURA DE LA MEMORIA

La memoria de este Proyecto Fin de Carrera se estructura de la siguiente forma.

1.- INTRODUCCIÓN
En primer lugar se introduce al lector al entorno donde se desarrolla el trabajo de investigación. Se hablará sobre los objetivos principales del proyecto.

2.- ESTADO DEL ARTE
Se comentan los distintos trabajos previos relacionados con el tema del proyecto. Esta parte sirve para poner en antecedentes al lector sobre cómo está el ambiente de trabajo con respecto a este tema en concreto.

3.- TRANSFORMADA SIFT
Tras un comentario sobre el escenario de trabajo, se explica de manera detallada la Transformada SIFT. Se expone cómo el algoritmo es capaz de extraer características invariantes a la escala, rotación e iluminación y cómo posteriormente es capaz de buscar coincidencias entre las características almacenadas y las extraídas de una imagen de entrada.

4.- DISEÑO E IMPLEMENTACIÓN
En este capítulo, se detalla el diseño que el sistema sigue y cómo se han llevado a cabo las distintas funciones y procesos que el programa es capaz de desarrollar.

5.- PRUEBAS Y CONCLUSIONES
Los test que se han realizado para comprobar el funcionamiento del programa se detallan en este apartado. También se realiza una explicación de los resultados obtenidos.

6.- TRABAJOS FUTUROS
Se comentan cuáles serían las aplicaciones y posibles adelantos de este programa en futuras investigaciones.

PRESUPUESTO
Si el programa formara parte de una actividad comercial este sería su presupuesto. En el caso de este proyecto, es un apartado totalmente informativo.

BIBLIOGRAFÍA
Contiene las referencias bibliográficas que se han consultado para realizar el proyecto.

ANEXO
Se añaden en este apartado los distintos archivos creados para el funcionamiento del sistema y el archivo de instrucciones de uso que se puede ver durante la ejecución del mismo.

2.- ESTADO DEL ARTE

2.1.- PROCESADO DE IMÁGENES

El procesado de imágenes o escenas visuales se remonta a los años 1920-1930 ([1] y [2]), cuando se intenta mejorar la calidad y disminuir el tamaño de las imágenes digitales transmitidas por un cable submarino por un periódico desde Londres hasta Nueva York. Este procesamiento hizo que la transmisión disminuyera su tiempo desde una semana a menos de tres horas.

Los problemas presentados en aquellos años al querer mejorar la calidad visual de las imágenes transmitidas estaban únicamente relacionados con la selección de los procesos de impresión y la distribución de los niveles de brillo de la imagen. En estos sistemas primitivos sólo eran capaces de codificar en cinco niveles de brillantez y no sería hasta 1929 cuando se incrementaran esos valores hasta 15 niveles.

A partir de entonces y durante más de 35 años se mejoraron los métodos de procesamiento, basándose únicamente en la capacidad de mejorar las transmisiones y posteriores reproducciones de las imágenes. Mediante la llegada de las primeras computadoras y programas de procesamiento, se observó el potencial del procesado digital de imágenes.

Este procesado de imágenes mediante programas de ordenador tuvo su inicio en el Jet Propulsion Laboratory, en Pasadena (California, EE.UU.) en 1964, cuando las imágenes de la luna fueron transmitidas por el Ranger 7 y procesadas por una computadora para corregir las diferentes distorsiones obtenidas en los sistemas de captura de las cámaras de televisión.

A partir de este hecho, el campo del procesamiento de imágenes ha experimentado un crecimiento acelerado, utilizando el procesamiento de imágenes para dos usos principales:

- Mejora de la calidad de la imagen y edición fotográfica.
  Mediante esta técnica, se puede realizar una adaptación de la calidad de la imagen digital para su posterior transmisión, almacenado, etc., y una modificación de la escena necesaria para su uso comercial, eliminación de residuos en la imagen, restauración de imágenes antiguas, etc. Con este tipo de procesamiento, se puede conseguir también avances en las aplicaciones médicas, arqueológicas, geográficas, etc.

- Reconocimiento de imágenes.
  El reconocimiento de características que contienen las imágenes han servido a lo largo de la historia de la robótica para realizar un reconocimiento del entorno que rodea al autómata; por otra parte se han utilizado estos procesos para realizar acciones que hasta entonces había hecho el ser humano, como el etiquetado de los contenidos de una escena.

2.2.- RECONOCIMIENTO DE IMÁGENES

El desarrollo de aplicaciones de procesamiento de imágenes ha dado un gran salto en las últimas décadas [3], debido a la aparición de ordenadores personales que permiten el uso de tecnologías multimedia y con una gran capacidad de cómputo para tratarlas. Será hace necesario poder encontrar una serie de patrones que proporcionen información relevante sobre el contenido de la imagen.

Un patrón es una característica con la que se representa una parte de la imagen para su posterior reconocimiento. El reconocimiento de patrones es la capacidad de generalizar a partir de observaciones. Esta capacidad es principalmente humana puesto que se relaciona con el reconocimiento o definición de un concepto.

Los sistemas de reconocimiento de patrones artificiales simulan esta habilidad mediante la creación y el uso de modelos físicos y matemáticos. Dada una aplicación concreta, sin embargo, un sistema artificial es preferible al uso de seres humanos debido a su velocidad, exactitud y robustez.

Existen diferentes enfoques a la hora de generar modelos matemáticos para el reconocimiento de patrones, siendo más populares los siguientes:

- Modelos Estadísticos.
  Estos modelos utilizan técnicas estadísticas y matemáticas, por lo que puede ser considerado como el enfoque clásico al reconocimiento de patrones aunque nuevas técnicas se siguen desarrollando en este campo.

- Redes Neuronales Artificiales (RNA).
  Siendo un enfoque más moderno, las RNA tienen varias décadas de historia. El fundamento de estos algoritmos consiste en la imitación del funcionamiento de las redes neuronales biológicas. La ventaja de usar redes neuronales está en el hecho que se pueden separar regiones no lineales de decisión tan complicadas como se desee dependiendo del número de neuronas y capas. Por lo tanto, las redes neuronales artificiales sirven para resolver problemas de clasificación de alta complejidad.

Muchos de los patrones que se pueden extraer de las imágenes, son variantes a cambios de orientación, escala, iluminación, etc, y por tanto no son útiles en reconocimientos de imágenes que puedan sufrir estos cambios. Es por esto, por lo que se realiza un reconocimiento mediante puntos clave invariantes.

El sistema utilizado en este proyecto fin de carrera es la Transformada SIFT, ideada por David Lowe en 1999 [11]. En ella, se intenta realizar una extracción de puntos clave (o keypoints) invariantes a la escala, orientación e iluminación, característicos de la escena que se esté tratando. La Transformada utilizará estos puntos clave para encontrar esos rasgos característicos en otras escenas diferentes. Este algoritmo ha sido ampliamente y utilizado e investigado, por lo que a continuación se realizará un recorrido por los avances más importantes del mundo del reconocimiento de imágenes mediante puntos clave invariantes.

2.3.- RECONOCIMIENTO DE IMÁGENES MEDIANTE PUNTOS CLAVE INVARIANTES

El procesado de imágenes mediante puntos clave de las mismas se remonta a 1981, cuando Moravec [4] realiza una búsqueda en estéreo de detección de esquinas en la imagen de referencia.

En esta investigación se habla sobre el control remoto de un autómata móvil. Mediante programa de ordenador se era capaz de guiar al robot a través de un camino con obstáculos, obteniendo imágenes a través de un sistema de televisión incorporado en el autómata, que utilizaba distintos tipos de localización estéreo para deducir el camino.

Tras calibrar el robot sobre una pared con diversos puntos de ajuste, se le colocaba en un punto del espacio de maniobras. Posteriormente se le proporcionaba una posición de destino y el sistema comenzaba el trabajo de reconocimiento y cálculos necesarios para evitar los obstáculos impuestos. El programa tomaba nueve nuevas imágenes para realizar el siguiente paso, y así continuamente hasta que el autómata alcanzaba a su destino u ocurría algún percance con alguno de los obstáculos del camino.

Los resultados obtenidos no fueron del todo satisfactorios, puesto que el sistema desarrollado sólo fue fiable para pequeñas distancias y con una velocidad muy lenta (en torno a un metro cada diez o quince minutos). No obstante, durante esta investigación se obtuvieron las bases del procesado de imágenes mediante puntos clave, ya que el sistema basa su reconocimiento del espacio mediante un visionado de imágenes proporcionados por el sistema de cámaras del autómata.

En este trabajo, se habla por primera vez de puntos clave o características de la imagen y se define el término característica como un punto en un espacio de tres dimensiones. Posteriormente dicha característica se podía localizar en distintos puntos de vista de la escena, aunque la elección de dichos puntos característicos era verdaderamente difícil. Por ejemplo una región de color uniforme no sería una buena característica puesto que sus partes son indistinguibles, pero se vislumbró que las esquinas con un gran contraste en direcciones ortogonales eran los mejores puntos clave que se podían localizar.
De esta forma comenzaron las primeras averiguaciones sobre las técnicas de localización de puntos clave.

La variación direccional de las características principales de las imágenes, se comenzó a medir en pequeñas ventanas cuadradas. Se calculaba la diferencia de cuadrados de los píxeles adyacentes en cada una de las cuatro direcciones (horizontal, vertical y diagonales) de cada ventana, escogiendo como característica de interés en dicha ventana el mínimo de estas cuatro sumas. Las características elegidas se almacenaban en una matriz ordenadas en orden decreciente para medir el interés de las regiones. Cada vez que se elegía una característica, se registraba su aspecto como una serie de fragmentos de la secuencia de imágenes reducidas. Se extraía una ventana 6x6 alrededor de la posición del punto característico de cada imagen reducida.

Como mucho se guardaba la imagen reducida a la mitad cuatro veces, 16 veces como la reducción a un cuarto, y así sucesivamente hasta que se obtuviera en alguno de los niveles la imagen al completo. El resultado final (como se observa en la imagen 2) era una serie de 6x6 imágenes, comenzando por una interpretación muy borrosa de la imagen completa y poco a

poco aumentando el zoom en las expansiones lineales de dos de los primeros planos de la característica.

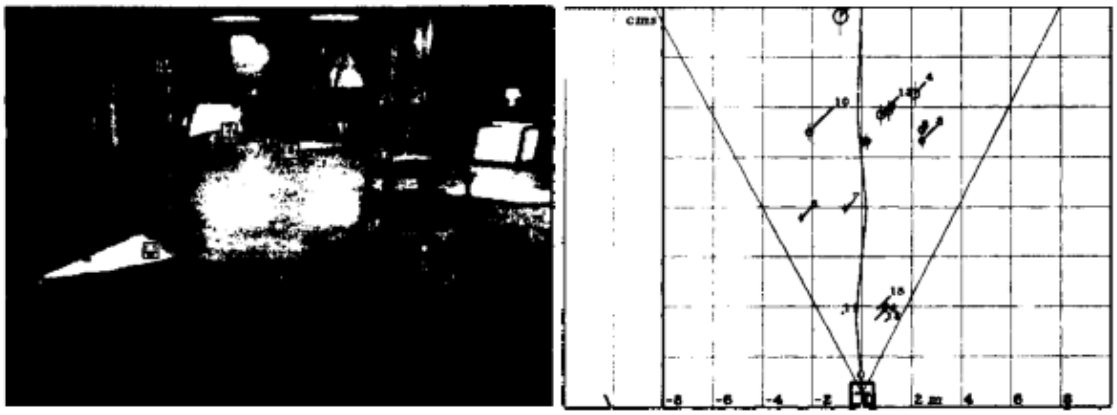


Imagen 2 - Visión de la habitación y puntos clave extraídos por el robot de Moravec.

Como debilidad principal, se obtiene que este método sólo era capaz de rechazar de forma inequívoca los bordes si estaban orientados a lo largo de los cuatro puntos cardinales de la suma. Los bordes con direcciones intermedias daban valores distintos a cero para las cuatro sumas y se elegían a veces incorrectamente como lugares interesantes de la imagen.

Este detector fue mejorado por Harris y Stephens en 1988 [5] haciéndolo más sensible a las pequeñas variaciones y a los bordes de los objetos contenidos en la figura procesada. Las conclusiones alcanzadas en esta investigación han sido utilizadas posteriormente para muchas otras tareas de comparación de imágenes.

Durante el trabajo de investigación se tuvo en cuenta que la consistencia del filtrado de los bordes de la imagen era primordial para la interpretación de secuencias de imágenes en tres dimensiones utilizando la función de algoritmos de seguimiento.

Se utilizó también la visión por ordenador para entender el mundo en 3 dimensiones sin ninguna restricción, en el que las escenas que se veían en general, contenían una amplia diversidad de objetos para realizar técnicas de trabajo "top-down"[1] en técnicas de reconocimiento.

Durante el desarrollo de la investigación de Harris y Stephens, también se trató de analizar el movimiento de unas secuencias monoculares de una cámara móvil. Mediante la extracción y el seguimiento de las características de las imágenes, las representaciones 3D análogas para dichas características podían ser construidas. Para realizar el seguimiento de las características de la imagen, debían ser discretas y no formar un área continua, como en texturas o bordes. Por eso, se realizaba una extracción y un seguimiento de los puntos o esquinas, ya que éstos son puntos interesantes en la imagen discretos y significativos.

[1] La técnica "top-down" [6] es una estrategia de procesamiento de información en el desarrollo de software. En un principio se formula un resumen del sistema sin especificar su contenido. Cada parte del sistema, es posteriormente diseñada con un mayor detalle. Una vez hecho esto con todos los módulos del programa, se redefinen con una mayor especificación hasta que dicha parte sea lo suficientemente detallada para validar el modelo.
Esta técnica se diseña frecuentemente con la ayuda de las denominadas "cajas negras", que hacen más fácil cumplir los requerimientos necesarios aunque no expliquen en detalle los componentes individuales contenidos en ellas.

Basándose en los errores cometidos por el diseño del detector de esquinas de Moravec, se descubrieron una serie de mejoras para una mejor detección de mejores puntos clave, con las que se realizó un reconocimiento de las características de las esquinas mucho más eficiente, y por lo tanto más sensible a los cambios de orientación:

- La respuesta del detector de Moravec era anisotrópica, ya que se tenían en cuenta sólo una pequeña porción de los cambios de orientación que podían darse. Esto se pudo mejorar mediante una realización del análisis sobre las ecuaciones del origen del desplazamiento, añadiendo los gradientes de los pequeños cambios a las ecuaciones iniciales utilizadas por Moravec.

- La respuesta de las esquinas proporcionada por Moravec era muy ruidosa porque la ventana utilizada era binaria y rectangular. En este caso, se utilizó una ventana de tipo Gaussiano para solucionar este problema.

- El operador respondía con demasiada facilidad a los bordes porque sólo se tenía en cuenta el mínimo de la suma de los cuadrados. Se reformuló la descripción de las esquinas para hacer uso de las variaciones que sufren las ecuaciones de desplazamiento con respecto a las direcciones del cambio.

Con estos cambios, se realiza un reconocimiento de las características de las esquinas mucho más eficiente, y por lo tanto más sensible a los cambios de orientación.

Años más tarde Harris [7] en 1993, demostró que el sistema desarrollado en 1988 desarrollado también servía para la detección del movimiento, en el que inicialmente se realizaba una detección de pequeños movimientos. En este trabajo de investigación posterior se hizo una extensión a problemas más complejos.

Zhang en 1995 [8] realizó una investigación en la que era posible, sobre una imagen de gran tamaño y usando el detector de esquinas de Harris, seleccionar los mejores puntos mediante una ventana de correlación cerca de cada una de las esquinas detectadas.
Las imágenes podían ser tomadas por una cámara sola en distintos instantes de tiempo o por distintas cámaras, puesto que no estaban calibradas.

Basándose en las técnicas clásicas de la detección de puntos clave (métodos de correlación y relajación) se tomaba un conjunto inicial de keypoints, y usando posteriormente una técnica robusta (menor mediana de los cuadrados) se descartaban los puntos clave no válidos. Se localizaron más puntos, mediante el uso de la geometría epipolar, utilizado también en la búsqueda estéreo.

Para la técnica de relajación, se definió una nueva medida de búsqueda, en la que se permitía una mayor tolerancia a la deformación de las transformaciones rígidas en el plano de la imagen y una menor contribución de los puntos clave más alejados.

Este algoritmo fue probado ampliamente y se observaron unos buenos resultados en escenas con muchos patrones repetitivos.

Al mismo tiempo Torr en 1995 [9], propuso hacer coincidir el movimiento a largo alcance, en el que las limitaciones geométricas se utilizaban para eliminar los valores extremos de los objetos en movimiento dentro de una imagen.

El trabajo que Schmid y Mohr realizaron en 1997 [10] demostró que las características locales

invariantes podrían ser extendidas a los problemas generales del reconocimiento de imágenes, en los que una característica se compara con una gran base de datos de imágenes (en este caso más de 1000).
Durante esta investigación también se utilizaron los detectores de esquinas de Harris para la selección de los puntos clave de las imágenes, pero en lugar de usar una ventana de correlación se usó un descriptor de rotación invariante por regiones en cada una de las imágenes. El sistema permitía los cambios aleatorios de orientación entre las dos imágenes (la de muestra y la de test).

Además se demostró que las características con muchas coincidencias podrían dar lugar al reconocimiento incluso con oclusión parcial y desorden en los objetos que aparecían en las imágenes.

El detector de esquinas de Harris es muy sensible a los cambios en la escala de la imagen, por lo que no proporcionaba una buena base para hacer coincidir imágenes con distintos tamaños.

Algunos trabajos de David Lowe anteriores a la creación del algoritmo SIFT [11] ampliaron las oportunidades de conseguir la invarianza ante la escala, centrándose más en características locales de las imágenes que se procesaban. En estas investigaciones también se describió un nuevo tipo de descriptores locales que proporcionaban más características relevantes, y a su vez eran menos sensibles a las distorsiones de las imágenes como por ejemplo los cambios de vista en 3D.

Con la llegada de la Transformada SIFT, se conseguirían también mejoras en la estabilidad y la invarianza de la función que realiza la extracción de los keypoints de la imagen.

Aparte de las investigaciones de David Lowe, existen múltiples trabajos de investigación anteriores al algoritmo SIFT, que fueron capaces de realizar identificaciones de puntos clave invariantes ante cambios de escala.

Uno de los primeros fue el desarrollado por Crowley y Parker en 1984 [12], que lo consiguieron representando los picos y valles de la escala en una estructura de tipo árbol. Esta estructura, podía ser compensada entre las imágenes con cambios de escala aleatoria. Las descripciones de las formas que estaban codificados en esta representación pueden ser encontradas eficientemente a pesar de los cambios de tamaño, orientación o posición.

En este trabajo se definieron en primer lugar las motivaciones para la resolución de una representación múltiple, seguida de la definición del algoritmo de denominada la "Transformada DOLP". A continuación se presentaron las técnicas para la codificación de una descripción estructural de las formas simbólicas de dicha transformación. El proceso consistía en la detección de picos y crestas locales en cada imagen paso banda y en todo el espacio tridimensional definido por la transformación DOLP. La vinculación entre los picos adyacentes de diferentes imágenes paso banda, daba lugar a un árbol de resolución múltiple que describía la forma. Los picos que a su vez eran máximos locales en dicho árbol, proporcionaban puntos de referencia para la alineación, la manipulación y la búsqueda de las distintas formas.

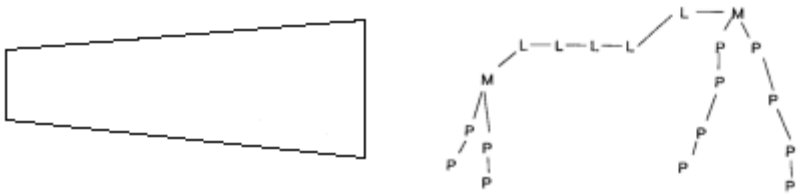


Imagen 3 - Figura romboidal y árbol DOLP asociado.

La detección y la vinculación de las crestas en cada imagen paso banda, proporcionaba un gráfico que unía los picos dentro de una forma y describía las posiciones de los límites de la forma en múltiples resoluciones. Dicha detección y vinculación de las crestas en el espacio del árbol DOLP se describía mediante formas alargadas y enlazaba los picos más altos del árbol, como puede observarse en la imagen 3.
También fueron descritos en este trabajo, los criterios para determinar la correspondencia entre los símbolos de las distintas descripciones. Tal búsqueda de correspondencias recibía una simplificación mediante el uso de las coincidencias a resoluciones más bajas.

Otro trabajo de investigación más reciente fue el realizado por Shokoufandeh, Marsic y Dickinson en 1999 [13], donde se proporcionaban más descriptores mediante coeficientes de la función wavelet.

Durante esta investigación, se introdujo un nuevo punto de vista basado en la representación del objeto, denominado "Mapa gráfico de prominencias" (*Saliency Map Graph,* SMG), que capturaba las regiones más destacadas de una vista de objetos a diferentes escalas utilizando una transformada wavelet, como se muestra en la imagen 4.

Esta representación compacta era muy invariante a la traslación, rotación (rotación en grados y de profundidad) y la escala, y ofrecía la descripción de los cuerpos necesaria para el reconocimiento de objetos ocluidos. Para comparar dos gráficos del mapa de prominencias, se presentaban dos algoritmos de similitud gráfica.

El primero calculaba la similitud topológica entre dos puntos, proporcionando un nivel de coincidencias a grandes rasgos en dos gráficos, y el segundo calculaba la similitud geométrica entre dos mapas, proporcionando unas coincidencias mucho mejores y más definidas.

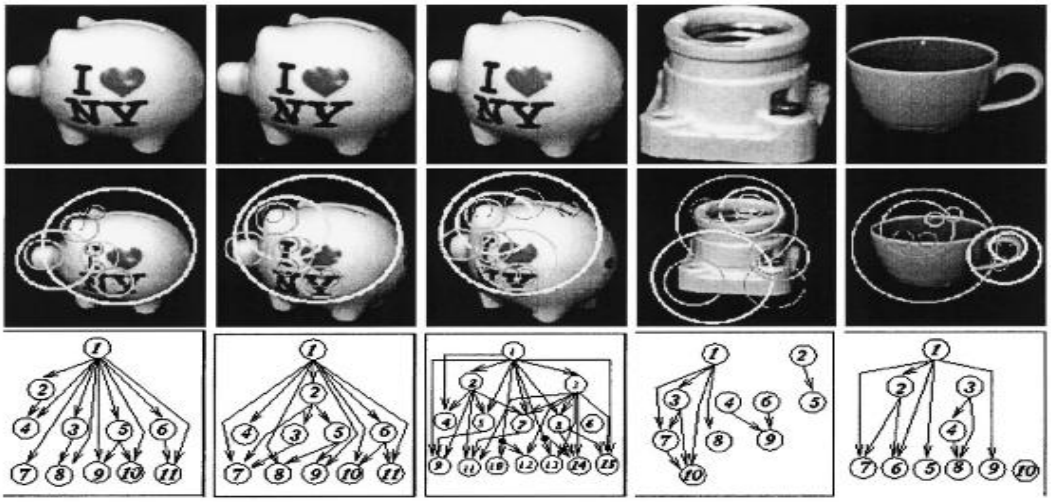


Imagen 4 - Distintas imágenes y sus mapas gráficos de prominencias.

En este trabajo se probaron y compararon ambos algoritmos mediante una gran base de datos de puntos de vista de distintos objetos.

El problema de la identificación de una escala adecuada y coherente para la detección de las características fue estudiado entre 1993 [14] y 1994 [15] por Lindeberg, planteándolo como un problema de selección de la escala adecuada.

En el primero de los trabajos de Lindeberg, se realizó una investigación sobre tres aspectos fundamentales:

- La representación multiescala de imágenes en escala de grises, lo que hizo explícitos los rasgos en la escala de espacio y las relaciones entre las estructuras a escalas diferentes.

- La metodología para la extracción de estructuras similares a la imagen de esta representación.

- Aplicaciones para la detección de bordes, análisis de histogramas, y la clasificación utilizada, demostrando cómo el método propuesto se podía utilizar para guiar los procesos visuales en una etapa posterior.

La representación, daba una descripción cualitativa de la estructura de la imagen, lo que permitía una detección estable de las escalas y las regiones de interés asociadas en un fondo único, utilizando una seguridad de los datos gestionados.

En otras palabras, se generaban señales de segmentación gruesa y por lo tanto, podía ser visto como un precedente de procesamiento adicional, que podía ser ajustado correctamente más tarde haciendo las señales de segmentación mucho más definidas. Se argumentó también que una vez que dicha información estaba disponible, muchas otras tareas de procesamiento podían realizarse de forma mucho más simple. Los experimentos con imágenes reales demostraron que la teoría propuesta daba resultados positivos y por tanto que las investigaciones realizadas podían ser un gran avance para el reconocimiento de imágenes.

En el segundo trabajo de Lindeberg, se explica cómo una propiedad inherente de los objetos en el mundo es que sólo existen como entidades significativas en ciertos rangos de escala. Si se trataba de describir mediante señales una estructura desconocida del mundo real, era de vital importancia obtener una representación multiescala de los datos.

En este artículo se ofrecía una revisión de un tipo especial de representación multiescala: la escala lineal del espacio de representación, que fue desarrollada por la comunidad de visión por computador para manejar estructuras de la imagen en diferentes escalas de una manera consistente. La idea básica es la de insertar la señal original en una familia con un parámetro de señales suavizadas poco a poco, en las que los detalles de escala fina eran sucesivamente suprimidos y se demostró que el núcleo de Gauss y sus derivados eran señalados durante las primeras etapas de dicha inserción.

Las condiciones que especifican el núcleo Gaussiano eran básicamente la linealidad y la invarianza al cambio, junto con las distintas maneras de formalizar la forma en que las estructuras a escala gruesa que debía corresponder a la simplificación de las estructuras correspondientes a escalas finas. Se debía poner atención a que esta señalización no fuera realizada de forma accidental, sino mediante el método de suavizado. Se observó también que

las distintas formas de elegir los axiomas del espacio de escala daban lugar a la misma conclusión.

La salida de la representación a escala del espacio podía ser utilizada para una variedad de principios de tareas visuales, clasificación de elementos y el cálculo de formas, y que se podían expresar directamente en términos de combinaciones (posiblemente no lineales) de los derivados de Gauss en múltiples escalas. En este sentido, la representación a escala del espacio podía servir como base para la visión artificial.

Durante las décadas anteriores a este trabajo de investigación se desarrollaron una serie de enfoques multiescala de representaciones, que estuvieron más o menos relacionados con la teoría de la escala espacio-temporal, en especial las teorías de las pirámides, olas y los métodos de multirred.

A pesar de sus diferencias cualitativas, la creciente popularidad de cada uno de estos enfoques indicaba que la idea fundamental de la escala era cada vez más apreciada por la comunidad la visión por computador y por los investigadores en otros campos relacionados. Una interesante similitud con la visión biológica es que los operadores del espacio escala se parecen mucho a los perfiles de campo receptivo registrado en los estudios neurofisiológicos de la retina de mamíferos y en la corteza visual. Por tanto Lindeberg, desarrolló estos dos estudios que más tarde serían utilizados para la realización del algoritmo SIFT

Muchos más trabajos fueron realizados para que las características locales fueran invariantes a las transformaciones que éstas pudieran sufrir. Baumberg en el año 2000 [16], realizó una investigación en la que se presentaba un método robusto de búsqueda de características de forma automática mediante las funciones de imágenes correspondientes a un mismo punto físico de un objeto visto desde dos puntos de vista arbitrario. A diferencia de los enfoques convencionales de búsqueda en estéreo, no se asumía ningún conocimiento previo sobre las posiciones de la cámara relativa y orientaciones. De hecho, en dicha aplicación la información se deseaba determinar a partir de las coincidencias de los puntos clave con los de la imagen de entrenamiento.

Las características eran detectadas en dos o más imágenes y caracterizadas mediante texturas afines invariantes. El problema de la ventana de efectos se trató explícitamente, donde la descripción de las características era invariante de las transformaciones lineales de los datos de imagen, como la rotación, la inclinación y el estiramiento. El proceso de comparación se optimizó para una aplicación de la estructura del movimiento que hiciera caso omiso de los partidos poco fiables a pesar de la reducción del número de coincidencias.

Tuyte-Laars y Van Goo durante el año 2000 [17] desarrollaron un trabajo donde se definió que las regiones invariables eran las partes de la imagen que se deformaban automáticamente con los cambios de puntos de vista para seguir cubriendo de forma idéntica las partes físicas de la escena que se estaban tratando. Dichas regiones eran descritas por un conjunto de rasgos invariantes, lo que hacía que fuera relativamente fácil buscar parecidos entre los distintos puntos de vista y bajo iluminaciones cambiantes.

En trabajos anteriores se habían presentado las regiones invariables basadas en una combinación de detección de esquinas y bordes. La aplicación discutida hasta entonces era la recuperación de la imagen de base de datos. En la investigación de Tuyte-Laars y Van Goo se presentó un método alternativo para la extracción de regiones invariantes, que no dependían de la presencia de bordes o esquinas de la imagen, sino que estaba puramente basado en la intensidad.

Además se demostró el uso de estas regiones para otra aplicación, la cual emparejaba todas las bases estéreo. El objetivo fue construir un sistema que explotaba varios tipos de regiones invariables como consideraba en cada momento. Esto brindaba más correspondencias y un sistema que podía hacer frente a una gama más amplia de imágenes, como se muestra en la imagen 5.

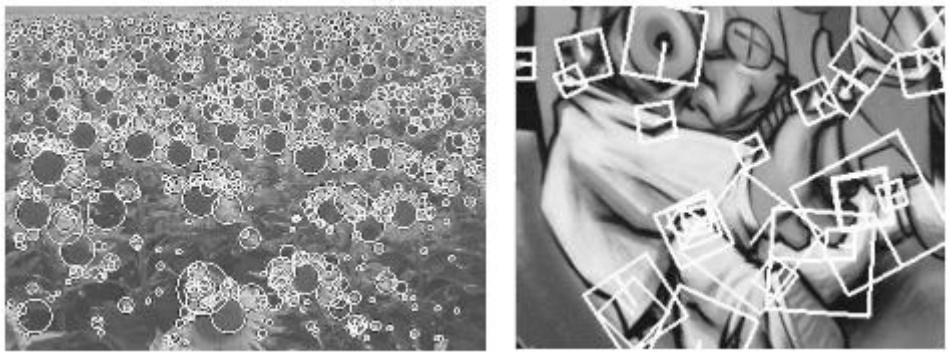


Imagen 5 - Puntos de interés en un campo de flores y distintos tamaños de las ventanas del descriptor en un graffiti.

Para aumentar la robustez del sistema aún más, se añadieron dos limitaciones semilocales (geométricas y fotométricas) en las regiones coincidentes, que permitían poner a prueba la consistencia de dichas coincidencias y por lo tanto rechazar las regiones que se habían emparejado de forma incorrecta.

Mikolajczyk y Schmid, en 2002 [18] presentaron en su artículo un nuevo enfoque para la detección de puntos de interés invariantes. Fueron capaces de hacer frente a importantes transformaciones incluyendo los grandes cambios de escala.

Tales transformaciones introdujeron cambios significativos en la ubicación, escala y forma de los vecinos de un punto de interés. El enfoque permitió resolver estos problemas simultáneamente.

Este algoritmo se basó en tres ideas clave:

- La “matriz de segundo momento” calculada en un punto se utilizó para normalizar una región de una manera invariante (inclinación y estiramiento).

- La escala de la estructura local se indicaba con extremos locales de normalización derivados de grandes escalas.

- Un detector adaptado de Harris determinaba la ubicación de los puntos de interés. Utilizando una versión multiescala de este detector para la inicialización del algoritmo.

A continuación un algoritmo iterativo modificaba la ubicación, escala y entorno de cada punto y convergía a puntos invariantes.

Para la búsqueda y el reconocimiento, la imagen se caracterizó mediante un conjunto de puntos invariantes. La transformación asociada a cada punto permitía el cálculo de un descriptor invariante que también era invariante a cambios de iluminación. Una comparación

cuantitativa del detector con los hasta entonces existentes mostró una mejora significativa en la presencia de grandes deformaciones.

Los resultados experimentales de búsquedas en bases de datos amplias mostraron un excelente desempeño en la presencia de grandes transformaciones de perspectiva y cambios de escala significativos.

Tras el considerable éxito logrado en la reconstrucción automática de secuencias de imágenes, donde los algoritmos de pequeñas referencias se podían utilizar para establecer coincidencias en una serie de imágenes Schaffalitzky y Zisserman deciden en 2002 [19], estudiar el caso de puntos de vista muy distantes entre sí, donde los métodos habían sido limitados a dos o tres vistas de las escenas tratadas. En dicho trabajo se investigó el problema del establecimiento de puntos de vista con respecto un gran número de imágenes proporcionadas en las que no se suministraba ninguna información.

Una aplicación típica sería aquella en la que las imágenes se obtienen de fuentes diferentes o en momentos diferentes, con distintos puntos de vista (posición, orientación y escala) y donde las condiciones de iluminación podían variar significativamente en el conjunto de datos. Aunque este algoritmo no era del todo idóneo para una búsqueda triple de referencias, ya que este proceso sería demasiado costoso al aumentar el número de puntos de vista tratados.

En su lugar se investigó cómo una combinación de las características invariantes de la imagen, puntos covariantes, y las múltiples relaciones entre los distintos puntos de vista podían ser utilizados conjuntamente para permitir búsquedas de coincidencias eficaces en puntos de vista múltiples.

El resultado fue un algoritmo de coincidencias lineal al número de puntos de vista. Los métodos se probaron en varios conjuntos de datos de imagen real con distintos puntos de vista. La investigación permitió encontrar una técnica de imagen basada para la navegación en una escena en 3 dimensiones, pasando de una escena a cualquier otra imagen siguiente más adecuada.

Brown y Lowe, también en 2002 [20] realizaron una investigación en la que se expuso el problema de encontrar correspondencias entre imágenes en las que hay grandes cambios en el punto de vista, la escala y la iluminación.

Los trabajos anteriores habían demostrado que los puntos de interés del espacio-escala podían encontrarse mediante la capacidad de repetición, a pesar de dichos cambios. Por otra parte, la alta entropía de las regiones de la imagen alrededor de descriptores locales significa que estas áreas eran altamente discriminatorias para las posibles coincidencias. Para los descriptores en los puntos de interés debía existir una búsqueda robusta entre imágenes y debía ser, en la medida de lo posible, invariante para el procesamiento de imágenes.

En dicha investigación se presentó una familia de funciones que utilizaban los grupos descriptores de regiones alrededor de los puntos de interés de la imagen de forma geométrica. Las características descriptoras estaban formadas mediante un muestreo de la relación de la imagen a los marcos canónicos definidos por los puntos.

Además de la búsqueda robusta, otra de las ventajas clave de este enfoque fue que cada coincidencia implicaba una hipótesis de una transformación local en 2D. Esto permitía también rechazar inmediatamente la mayor parte de las falsas coincidencias obtenidas utilizando una transformada de Hough. Se rechazaban también los valores extremos restantes utilizando

RANSAC y la restricción epipolar. Los resultados mostraron que la adecuación de función densa se podía lograr en unos pocos segundos utilizando ordenadores Pentium III a 1 GHz.

Todo esto permitió que la búsqueda de características invariantes en una superficie plana en proyecciones 3D se realizara en la mayor parte de los casos, mediante un remuestreo de la imagen en las regiones donde se quería afinar la búsqueda. Sin embargo ninguno de esos enfoques eran totalmente invariantes, ya que al comenzar con características de escala y localizaciones seleccionadas no era posteriormente posible, debido a su gran coste computacional, explorar todo el espacio completo de la imagen.

Otros muchos tipos de características fueron propuestas para su uso en el reconocimiento, algunos de los cuales se podrían utilizar junto con las funciones desarrolladas por David Lowe para proporcionar más coincidencias en distintas circunstancias.

Una de estas características es la que hace uso de los contornos de la imagen o los límites de la región que contiene el objeto, pero en este caso se debería tener en cuenta el fondo que lo rodeaba. Matas demostró en 2002 [21] que sus regiones máximamente estables pueden producir una gran cantidad de características coincidentes con una gran estabilidad.

Durante su investigación estudió el problema de base ancha en estéreo es decir, el problema de establecer correspondencias entre un par de imágenes tomadas desde diferentes puntos de vista. Se expuso un nuevo conjunto de correspondencias entre elementos de la imagen y se introdujeron unas nuevas regiones denominadas “extremas”.

Dichas regiones extremas poseen unas propiedades muy buenas, donde el conjunto de estas áreas era cerrado bajo:

- Transformación continúa de coordenadas de la imagen.

- Transformación monótona de las intensidades de la imagen.

- Se presentó también una detección eficaz (con gran complejidad lineal) y un algoritmo rápido (cerca de la velocidad de los fotogramas de un vídeo) para un subconjunto estable e invariante de las regiones extremas: las “Regiones Extremas de Máxima Estabilidad” (*Maximally stable extremal regions*, MSER).

Se propuso a su vez una nueva medida de similitud sólida para el establecimiento de correspondencias entre imágenes. La robustez se aseguraba de que las múltiples regiones invariantes de medición (regiones obtenidas por las construcciones invariantes de regiones extremas), eran significativamente más grandes (y por tanto discriminatorias) que las MSER y podían ser utilizadas para buscar coincidencias.

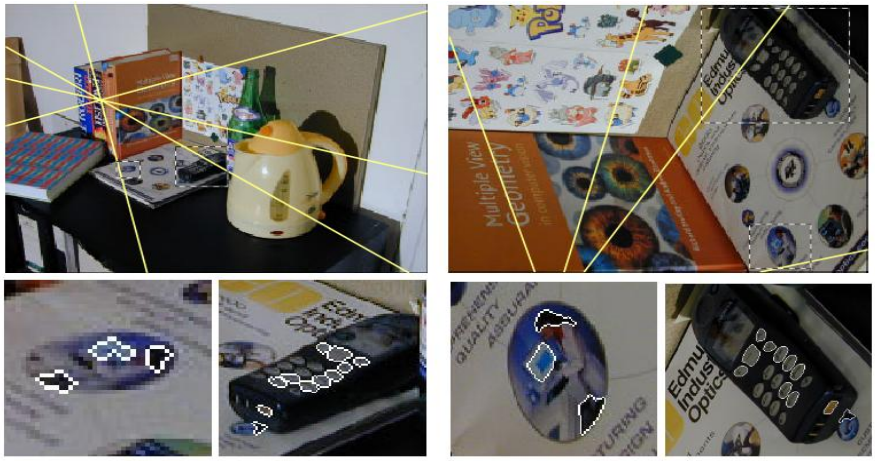


Imagen 6 - Detección de regiones MSER en cambios de orientación y escala.

La gran utilidad de las regiones MSER, al escoger varias de estas regiones de medición y una métrica robusta se demostró también en esta investigación utilizando grandes bases de pares de imágenes de distintas escenas (interiores y exteriores). Estos experimentos se realizaron con cambios de escala (en un factor 3,5), de las condiciones de iluminación, rotando la imagen fuera del plano, utilizando oclusión, cambios de escala local anisotrópicos y la traducción en 3D de los puntos de vista presentes en los problemas de prueba. De esta forma, se obtuvieron unas estimaciones muy buenas de la geometría epipolar (distancia media de los correspondientes puntos a la línea epipolar por debajo de los 0,09 de distancia entre píxeles).

Mikolajczyk en 2003 [22] desarrolló un nuevo descriptor que usaba bordes locales sin tener en cuenta los bordes cercanos que no pertenecen al objeto a reconocer, proporcionando la capacidad de encontrar las características más estables incluso cerca de formas estrechas superpuestas y desordenadas del fondo.

En dicho trabajo se describió un método para el reconocimiento de objetos con poca textura que podían contener agujeros y partes tubulares en escenas con condiciones de visión desordenadas arbitrariamente. Con este fin se desarrollaron una serie de nuevos componentes.

En primer lugar, se introdujo un nuevo límite en función del detector local invariante a las transformaciones de similitud. Las características se localizaban en los bordes y se estimó de manera invariable la escala en los vecinos más cercanos del punto. En segundo lugar, el descriptor local calculado para las características del plano no se veía afectado por el desorden del fondo, incluso cuando la función está en un límite de objeto. En tercer lugar, el descriptor se generalizó mediante método SIFT de Lowe para tener en cuenta los bordes.

Se realizó un entrenamiento para que el sistema aprendiera el modelo de objetos. El objetivo propuesto entonces fue reconocer en las nuevas imágenes de una serie de medidas sobre las que se aplicaron progresivamente mayores restricciones geométricas. La última aportación de este trabajo fue la de conseguir una flexibilidad suficiente en la representación geométrica de los objetos de la clase visual que podían ser reconocidos.

Nelson y Selinger en 1998 [23] consiguieron unos buenos resultados con las características locales basándose en las agrupaciones que se concentran en los contornos de la imagen.

En dicha investigación, se describieron varias pruebas a gran escala del desempeño de un buen rendimiento para el reconocimiento de esferas y semiesferas de hasta 24 objetos complejos, con curvas, robustez contra el desorden y oclusión y algunos comportamientos interesantes reconocimiento genérico. También se estableció un protocolo que permitía, en presencia de cantidades cuantificables de desorden y oclusión, predecir sobre la base de las estadísticas de índice sencillo derivadas de las imágenes de ensayo limpio y puro de desorden de imágenes.

Del mismo modo Pope y Lowe en el año 2000 [24] dieron uso a las características basadas en la agrupación jerárquica de los contornos de la imagen, que eran particularmente útiles para los objetos con texturas detalladas. Este trabajo vuelve a basarse de nuevo en la utilidad de obtener unas características de unas imágenes de entrenamiento para el reconocimiento posterior de los objetos de las imágenes de test.

El modelo utilizaba las distribuciones de probabilidad para describir la gama de posibles variaciones en la apariencia del objeto. Estas distribuciones se organizaron en dos niveles.

Las grandes variaciones están se obtenían de la partición de las imágenes del entrenamiento en grupos correspondientes a los distintos puntos de vista del objeto. Dentro de cada grupo, las variaciones más pequeñas se representaban mediante distribuciones caracterizadas por la incertidumbre en la presencia, posición, y las mediciones de los distintos elementos discretos de apariencia.
Se utilizaron muchos tipos de funciones, que fueron desde la abstracción a partir de segmentos de borde hasta los grupos de percepción y de las regiones.

Un procedimiento de igualación utiliza la información sobre la característica de incertidumbre para orientar la búsqueda de una coincidencia entre el modelo y la imagen. La hipótesis de pares de funciones se utilizó para estimar las transformaciones, teniendo en cuenta el punto de vista de la incertidumbre característica. Estos métodos se implementaron en un sistema de reconocimiento de objetos, OLIVER. Mediante diversos experimentos se demostró que dicho sistema era capaz de aprender a reconocer objetos complejos en las imágenes desordenadas, mientras que la adquisición de modelos que representaban los objetos con puntos de vista distintos fueron relativamente pocos.

La historia de la investigación para el trabajo de reconocimiento visual alberga un gran conjunto de propiedades de la imagen que pueden ser utilizadas para realizar dicho trabajo.

Carneiro y Jepson en 2002 [25] describieron las características locales basadas en la eliminación que tienen más en cuenta la fase que las frecuencias espaciales locales, lo que mejora la invariancia ante distintos grados de iluminación.

En este trabajo se introdujo un nuevo tipo de característica local basado en la fase y en la amplitud de respuestas de los valores complejos de filtros orientables. El diseño de esta característica local fue motivada por el deseo de obtener vectores de características que fueran semiinvariantes bajo deformaciones comunes de imágenes, pero lo suficientemente distintivos como para proporcionar información de la identidad útil.

Una de las propuestas de tales características locales consistía en la combinación diferencial invariante a deformaciones de la imagen, tales como la rotación, como puede observarse en la imagen 7.

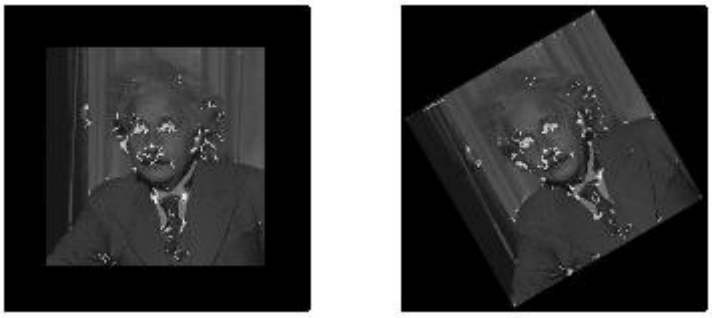


Imagen 7 - Puntos clave en la imagen original y tras una rotación de 30 grados.

El enfoque de Carneiro y Jepson se diferencia en que se consideró una clase más amplia de deformaciones de la imagen, incluyendo la adición de ruido, junto con las variaciones de brillo global y local; usando filtros direccionales para hacer la función robusta a la rotación. Se explotó el hecho de que los datos de fase es a menudo localmente estable con respecto a los cambios de escala, el ruido y los cambios frecuentes de brillo.

Por último se ofrecieron diversos resultados empíricos comparar dicha característica local con una basada en los invariantes diferenciales. Los resultados mostraron que la fase basada en las características locales lleva a un mejor rendimiento cuando se trata de la iluminación común con cambios y rotación 2D, al tiempo que los efectos comparables en términos de cambios de escala.

Schiele y Crowley en el año 2000 [26] propusieron el uso de histogramas multidimensionales. Este tipo de características son particularmente útiles para el reconocimiento de objetos con falta de detalles.

La aparición de un objeto se compone de la estructura local. Esta estructura local puede ser descrita y caracterizada por un vector de características locales medido por los operadores locales, tales como los derivados de Gauss o los filtros de Gabor.
En el artículo de Schiele y Crowley se presentó una técnica donde las apariencias de los objetos estaban representadas por las estadísticas conjuntas de los operadores vecinos. Por lo tanto, esto simbolizaba una nueva clase de técnicas de aspecto según la visión por computador.

Con base en estadísticas conjuntas, el trabajo desarrolla las técnicas para la identificación de varios objetos en posiciones arbitrarias y orientaciones en una escena desordenada. Los experimentos demostraron que esta técnica podía identificar la presencia de más de 100 objetos con obstrucciones importantes.
Lo más notable, es que esta técnica tiene baja complejidad, por lo que se ejecuta en tiempo real.

Basri y Jacobs en 1997 [27] demostraron el valor de la extracción de las fronteras de las regiones locales para su posterior reconocimiento. En este trabajo se comenta que los sistemas de reconocimiento intentan recuperar la información sobre la identidad de los objetos observados y su ubicación en el medio ambiente.

Un problema fundamental que se plantea es en reconocimiento es la estimación. Este es el problema de la utilización de una correspondencia entre algunas partes de un modelo de objetos y algunas partes de una imagen para determinar si la imagen contiene una instancia

del objeto y en caso de que lo haga, para determinar la transformación que se relaciona con el modelo de la imagen.

Los enfoques hasta 1997 para este problema se dividían en métodos que utilizaban las propiedades globales del objeto (por ejemplo, su centro de gravedad y momentos de inercia) y los métodos que utilizan propiedades locales del objeto (por ejemplo, esquinas y segmentos de línea). Las propiedades globales son sensibles a la oclusión y en concreto, a la oclusión de uno mismo; estas características son difíciles de localizar de forma fiable, y su búsqueda consiste en el cálculo intensivo.

Basri y Jacobs presentaron un nuevo método para el reconocimiento que utilizaba la información de la región. En este enfoque la imagen se divide en regiones. Teniendo en cuenta el partido entre los subgrupos de las regiones (sin ningún tipo de correspondencia explícita entre ellas), se calculaba la transformación de alineación.

El método se aplicaba a objetos planos similares, y las transformaciones proyectivas y las proyecciones de objetos 3-D sometidas a transformaciones proyectivas. Se combinaron muchas de las ventajas de los dos enfoques, evitando al mismo tiempo algunas de sus fallos.
Al igual que los métodos globales, dicho enfoque hacía uso de la información de la región que refleja la verdadera forma del objeto, y al igual que en los métodos locales, se podía manejar la oclusión.

Otras características útiles para incorporar a los sistemas de reconocimiento son el color, el movimiento, la discriminación del fondo, los descriptores de las distintas regiones, formas y profundidades de las señales estéreo.

Los programas ya desarrollados deberían poder incorporar fácilmente las nuevas características, ya que con ellas podrían dar solidez para proporcionar muchos más resultados correctos sin añadir un coste computacional demasiado elevado.

Mediante todas estas investigaciones en el trabajo que David Lowe escribe en el año 1999 [11], se habla sobre el reconocimiento de objetos mediante este tipo de características invariantes a la escala. A su vez, desarrolla un algoritmo capaz de encontrar estas características en imágenes de referencia y posteriormente utilizarlos para reconocer los objetos que aparecen en otras imágenes. De esta forma se obtiene un paquete de algoritmos que será usado para algunas aplicaciones que requieran del reconocimiento de objetos para su correcto funcionamiento. Son tantas las acciones que podrían realizarse con la Transformada SIFT que varias compañías han desarrollado agrupaciones de estos algoritmos para los distintos lenguajes de programación.

2.4.- PAQUETES DE FUNCIONES SIFT

En los trabajos de investigación redactados por David Lowe en 2001 [28] y en 2004 [29], se sientan las bases para el reconocimiento de imágenes mediante la transformada SIFT.

A partir de aquí, el paquete de funciones de SIFT crece a muchos lenguajes de programación y sistemas operativos, para su uso en diversos escenarios. David Lowe hace pública en su página web [30] una demo inicial de su programa en Junio de 2003, a la que seguirán otras tres versiones a lo largo de los años 2004 y 2005.
En la última de estas versiones, se realiza una mejora en la implementación del código para

Matlab en la que se incluyen nuevas formas de localizar coincidencias entre los puntos clave de la imagen de referencia y los de las nuevas imágenes.

Desde este momento, diversas empresas y universidades crean paquetes de funciones para distintos lenguajes y los cuelgan en la red para el uso y disfrute de todos los usuarios que deseen utilizarlos para sus investigaciones.

IPOL[2] es una comunidad, con sede en Francia, que desarrolla aplicaciones basándose en el procesamiento de imágenes y posteriormente las publica en su sitio web para su libre uso. Una de las aplicaciones generadas por esta empresa es la denominada VL_FEAT, iniciada por Andrea Vedaldi y Brian Fulkerson en 2007 [31]. Esta aplicación será la que se use en el desarrollo de este proyecto.

Los dos creadores de este paquete de utilidades se basaron en sus anteriores investigaciones sobre el reconocimiento de imágenes y en 2007 finalizaron la creación de una nueva biblioteca de funciones SIFT para los lenguajes C y Matlab (para todos los sistemas operativos).
Actualmente, la librería se encuentra en su versión 0.9.9 y fue alojada en la web en Junio de 2010. La descarga de este toolbox es totalmente gratuita y libre, y por tanto será el paquete que utilice en el desarrollo del sistema de reconocimiento. Este conjunto de funciones permite la extracción de las características SIFT de una imagen y su posterior uso para el reconocimiento de los objetos contenidos en esas imágenes y que podrían estar en otras imágenes.

A medida que se ha dado la posibilidad de descargar paquetes de algoritmos SIFT abiertamente al mundo de la investigación, son muchas las compañías e investigadores que han utilizado estas herramientas para desarrollar distintas aplicaciones relacionadas con el reconocimiento de imágenes.

2.5.- APLICACIONES DESARROLLADAS A PARTIR DE SIFT

Son muchas las aplicaciones desarrolladas desde que en Junio de 2003 David Lowe hiciera pública la demo de un sistema basado en la Transformada SIFT. Este hecho, unido con la gran cantidad de lenguajes que implementan el algoritmo y añaden métodos de búsqueda y emparejamiento entre keypoints, hacen que el uso de esta Transformada esté cada vez más extendido.

Uno de los primeros trabajos de investigación desarrollados a partir de la publicación de los algoritmos de Lowe, es el descrito en 2004 por Tony Lindeberg y Lars Bretzner [32];En él se describe cómo realizar el cálculo de los descriptores en tiempo real y en un ordenador normal.
Para ello trabajaron con representaciones multiescalas híbridas, en las que se pretendían calcular variables que equilibraran las ventajas de usar las pirámides y la representación espacio-escala para obtener una mayor eficiencia computacional y precisión del cálculo.

También aportaron un nuevo uso para el algoritmo SIFT, donde se describe la posible utilización de esta transformada en el reconocimiento de patrones de videos en movimiento.
Siguiendo con la línea de investigación de la localización de puntos de interés en el espacio-tiempo, se definieron varios tipos de descriptores de imagen a través de puntos locales en zonas de la imagen. Tras desarrollar el código necesario y probar con el movimiento de

[2] http://www.ipol.im/

aviones, se pudo lograr una clara ventaja del uso de este algoritmo frente a las acciones humanas realizadas en los mismos vídeos y almacenadas en una base de datos.

El propio Lowe en ese mismo año [33], desarrolla una investigación sobre la construcción automática de panorámicas. En este caso, se utilizan técnicas de reconocimiento basadas en características invariantes locales para observar qué imágenes se corresponden entre sí y un modelo probabilístico para su posterior verificación. Debido a estas técnicas el método es insensible al orden, orientación, escala e iluminación de las imágenes. Como también es sensible al ruido de las imágenes que no forman parte del paisaje en absoluto, es posible por tanto reconocer dichos lugares.

El trabajo desarrollado también en 2004 por Svetlana Lazebnik, Cordelia Schmid y Jean Ponce [34], habla sobre la búsqueda de partes invariantemente geométricas y expresivas de objetos 3D. Estos grupos denominados semi-local afines, se aprenden con la búsqueda entre los pares de imágenes de entrada no segmentados y desordenados, seguido por la validación con imágenes de entrenamiento adicionales. El enfoque propuesto se aplica al reconocimiento de mariposas. En la imagen 8 se aprecian algunas de las imágenes utilizadas en este trabajo.

Imagen 8 - Base de datos de mariposas del trabajo de Svetlana Lazebnik, Cordelia Schmid y Jean Ponce.

Uno de los trabajos más importantes llevados a cabo durante ese año es el descrito por Yan Ke y Rahul Sukthankar [35], en el que se describe la denominada PCA-SIFT. Este nuevo algoritmo mejora los descriptores locales de las imágenes utilizados por SIFT. En lugar de utilizar un suavizado de histogramas ponderados, se realiza un análisis de los componentes principales (PCA), para la normalización del gradiente. En experimentos desarrollados durante esta investigación, se demostró que los descriptores locales eran más distintivos, compactos y robustos para las deformaciones de las imágenes que los descriptores de representación SIFT.

Durante el año 2005 se realizaron multitud de investigaciones sobre la tecnología SIFT, aunque el trabajo más significativo es el efectuado por Krystian Mikolajczyk y Cordelia Schmid [36] en el que se describe una evaluación de desempeño de los descriptores locales SIFT. Se realizó una comparación de todos los descriptores existentes hasta la fecha y se propuso una extensión del descriptor SIFT que posteriormente demuestró ser mejor que el método original.

Es en 2006 cuando Herbert Bay, Tinne Tuytelaars y Luc Van Gool [37] desarrollan un nuevo tipo de características SIFT denominadas SURF (Speeded Up Robust Features) igual de buenas (o mejores) que las anteriores respecto a la distinción, solidez, etc., pero mucho más rápidas. Este hecho se logra recurriendo a la integral de las circunvoluciones de la imagen, aprovechando así los puntos fuertes de los detectores principales existentes y los descriptores

(usando una matriz de base Hesse medida por el detector y basada en un descriptor de distribución), y mediante la simplificación de éstos métodos esenciales.

David Lowe también pone su grano de arena durante ese año, escribiendo una investigación junto a Iryna Gordon [38] sobre la aplicación del reconocimiento de objetos en 3D. Proponen un modelo métrico en 3D a partir de varias imágenes tomadas sin calibrar la cámara. El reconocimiento de estos modelos en las nuevas imágenes y resolver con precisión esos objetos son el objetivos de este proyecto.

Ivan Laptev, Barbara Caputo, Christian Schuldt y Tony Lindeberg [39], desarrollan en 2007 un trabajo sobre el problema del reconocimiento de movimiento con eventos basados en representaciones de movimiento locales. Se supone que los patrones de movimiento similares, contienen eventos similares con el movimiento consistente a través de secuencias de imágenes, aunque se acaba demostrando que utilizando una base de datos de acciones humanas se obtienen unos resultados similares que con el algoritmo desarrollado.

Es también durante ese año cuando Paul Scovanner, Saad Ali y Mubarak Shah [40] investigan sobre la descripción 3D del descriptor SIFT de video o imágenes en 3 dimensiones, como por ejemplo los datos de resonancias magnéticas. Se demuestra cómo este nuevo método es capaz de superar a los utilizados anteriormente de una manera más eficiente. Este algoritmo usa una bolsa de palabras para representar vídeos y descubrir las relaciones espacio-temporales de las palabras para describir mejor los datos contenidos en el vídeo.

La primera vez que se relacionan las proyecciones invariantes de 3D a 2D de conjuntos de puntos es durante el año 2008 por YuanBin Wang, Zhang Bin, Yu Ge [41]. La información de la profundidad se pierde durante la proyección de un objeto en 3D a una imagen 2D. Sin embargo, hay relaciones invariantes sobre las proyecciones de un conjunto 3D y las de su imagen 2D. En esta investigación se presenta un método general para calcular esas relaciones.

Durante ese mismo año, un grupo de investigadores (Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond, Dieter Schmalstieg) [42] desarrolla un planteamiento del seguimiento de las características naturales en teléfonos móviles.
En ese trabajo se presentan dos técnicas para el seguimiento natural en tiempo real en teléfonos móviles, lográndose velocidades de cuadro de hasta 20Hz para la función natural del seguimento de objetos con texturas planas en la generación de teléfonos que en esa época existían.

Para lograr este desarrollo, se utilizó un enfoque basado en la técnica de los descriptores SIFT, combinándola con técnicas Ferns, que requiere grandes cantidades de memoria, aunque es mucho más rápido. Tras el desarrollo, se presentan evaluaciones de robustez y rendimiento de varios dispositivos y finalmente se discute el uso de este algoritmo para aplicaciones de realidad aumentada.

En 2009, se realiza un desarrollo de aplicaciones SIFT con histogramas de orientación irregular. Este trabajo es desarrollado por un grupo de investigadores (Yan Cui, Nils Hasler, Thorsten Thormahlen, Hans-Peter Seidel) [43]. En él se comenta cómo nunca se puede conseguir la perfecta invariancia en la escala debido al muestreo, ruido en los datos de las imágenes, los límites computacionales, etc. En esta ocasión se desarrolla un enfoque irregular de la red, que reduce los efectos negativos del error en la escala y aumenta la precisión de las características de la imagen. Como resultado se muestra un conjunto estándar de datos que muestran que el enfoque de la red irregular supera los descriptores SIFT originales.

También durante ese año, se realizan diversos Proyectos Fin de Carrera en la Universidad Carlos III de Madrid cuyas investigaciones se basan en el algoritmo SIFT. Uno de ellos es el desarrollado por Sara Pardo Feijoo [44], cuya lectura se realizó en Julio. El proyecto trata sobre el modelo "*Bag of words*" (*BoW*) aplicado al reconocimiento y clasificación de tres grupos distintos de imágenes. Este modelo se basa en la inclusión de *palabras clave* de la imagen en un álbum de palabras, que más tarde servirán para reconocer las coincidencias entre dos imágenes consecutivas de una secuencia de vídeo y así poder reconocer la velocidad y dirección de un objeto cualquiera. Este método se utiliza sobre todo en visión artificial. Se investiga la realización del reconocimiento mediante el agrupamiento con el algoritmo *K-means* y la clasificación mediante las redes neuronales y las máquinas de vectores de soporte (*SVM*).

Durante el desarrollo de ese mismo año, se hace la lectura de otro proyecto desarrollado por Juan Manuel Peraza Domínguez [45], que trata de realizar un algoritmo para la estimación de la distancia recorrida por un robot móvil mediante la utilización de los descriptores SURF. En este trabajo, se hace una investigación de las distintas opciones de SIFT y SURF y una comparativa del uso de estos dos algoritmos frente al problema presentado.

Francesc Tarrés González de la Universidad Politécnica de Cataluña [46], realiza también en el transcurso de 2009. En él, se realiza la investigación del método de detección de esquinas de Harris y del algoritmo SIFT y se aplican los resultados a un trabajo de detección de rasgos faciales.

Durante 2010 se realizó un trabajo de aplicación del algoritmo SIFT a la medicina, utilizando este algoritmo para el reconocimiento de patrones de Alzheimer en imágenes del cerebro (tomadas de la base de datos libre OASIS). Esta investigación, desarrollada por varias universidades de Estados Unidos y Canadá [47], pretende dar un nuevo enfoque a las prácticas de la transformada SIFT, para que no solamente sirvan para el reconocimiento de imágenes, sino que se puedan aplicar de alguna forma para realizar programas médicos fiables.

También durante 2010 Greg Flitton, Toby P. Breckon y Najla Megherbi [48], realizaron una investigación donde aplicaron el reconocimiento SIFT en 3D a objetos con volúmenes complejos. Este tipo de reconocimiento se usa en la seguridad de la aviación y por tanto es un buen camino para desarrollar nuevos algoritmos y programas que permitan un mejor reconocimiento.

Se desarrolla también un Proyecto Fin de Carrera en la Universidad de Murcia por Arturo Nicolás Pina [49], en el que se investiga e implementa un sistema de clasificación y búsqueda automática de imágenes no segmentadas. Para realizar esta aplicación se utilizan técnicas de comparación basadas en color y en puntos característicos (tanto SIFT como SURF).

Como se puede apreciar son muchas y variadas las utilidades de los paquetes de herramientas SIFT. Es por esto, por lo que el campo de investigación de este algoritmo es tan extenso y útil para desarrollar nuevas aplicaciones.

3.- ALGORITMO SIFT

En este apartado, se describirá cómo el algoritmo SIFT es capaz de extraer las características y formar los keypoints (puntos clave). Ante la imposibilidad de conseguir una descripción en profundidad del algoritmo, se ha recurrido a traducir y adaptar uno de los documentos de investigación de David Lowe escrito en 2004, y llamado "Distinctive image features from Scale-Invariant Keypoints" [29], donde se exponen de manera detallada los pasos necesarios para extraer las características invariantes a la escala, orientación e iluminación procedentes de la transformada SIFT.

El objetivo principal del algoritmo SIFT es la extracción de unas características que describan de forma correcta los objetos que aparecen en las distintas imágenes que se tengan recopiladas. De esta forma se podrán observar las similitudes existentes entre las imágenes almacenadas en la base de datos y la imagen introducida al programa para su reconocimiento.
El coste de la extracción de dichos descriptores debe ser el mínimo posible, y esto se consigue mediante un filtrado en cascada, en los que los algoritmos más pesados sólo se realizan en los lugares que han pasado los filtrados anteriores.

Los distintos filtrados que van atravesando las características son los siguientes (de menor a mayor coste):

- Detección de extremos en la escala: es la primera de las etapas, en la que se buscan todas las posibles escalas y localizaciones en la imagen. Mediante la función de la Diferencia de Gaussianas es posible realizar esta tarea de forma eficiente, y así identificar los keypoints potenciales invariantes a la escala y orientación.

- Localización de los keypoints: los puntos clave se eligen en base a las medidas de estabilidad. Estos keypoints almacenarán el lugar, la orientación y la escala.

- Asignación de la orientación: a cada punto clave se le asigna una o varias orientaciones basándose en las direcciones locales de la imagen gradiente, es decir en las variaciones de píxeles cercanos en las dos dimensiones.

- Descriptor del keypoint: los gradientes locales de la imagen se miden en la región que rodea al punto clave. Éstos son transformados mediante una representación que permitirá medir niveles de distorsión y cambios en la iluminación de forma local.

3.1.- DETECCIÓN DE EXTREMOS EN LA ESCALA-ESPACIO

En este primer paso, es necesario detectar una serie de puntos clave mediante una serie de algoritmos en cascada. Estos algoritmos, irán filtrando los puntos de la imagen que más posibilidades tengan de llegar a ser un keypoint, para realizar posteriormente en torno a ellos un examen más exhaustivo.

En un primer momento, se deben localizar los lugares y escalas que pueden repetirse en distintas vistas del objeto. La localización de estos lugares puede hacerse mediante una búsqueda de características estables en todas las escalas que se den en la imagen, utilizando una función continua conocida como "Escala-espacio", desarrollada en 1983 por Witkin y descrita en su trabajo de investigación en 1986 [46].

Koenderink en 1984 [50] y Linderberg en 1994 [15] demostraron que bajo algunas circunstancias, la única posibilidad de utilizar una función del tipo "Escala-Espacio" es mediante la función Gaussiana. Por lo tanto, la función que define la "Escala-Espacio" de una imagen sería del tipo $L(x, y, \sigma)$.

Esta función se produce tras la convolución de la variable de escala Gaussiana $G(x, y, \sigma)$ y una imagen de entrada $I(x, y)$:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y); \quad (1)$$

donde $*$ es la operación de convolución en *x* e y; y:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \nabla \quad (2)$$

Para detectar eficientemente las posiciones de los puntos clave, Lowe propuso en 1999 [11] una nueva forma de hacerlo, utilizando la detección de extremos de la "Escala-Espacio" en una función de convolución de Diferencia de Gaussianas con la imagen:

$$D(x, y, \sigma) = \big(G(x, y, k\sigma) - G(x, y, \sigma) * I(x, y)\big) = L(x, y, k\sigma) - L(x, y, \sigma) \quad (3)$$

Como el propio Lowe explica, existen varias razones para escoger esta función:

- En primer lugar es una función particularmente eficiente. Las imágenes suavizadas, L, necesitan ser tratadas en algunos casos con una descripción de las características de espacio y escala conllevando este paso una gran carga computacional. La ecuación de D(x,y,σ) puede ser calculada, mediante esta forma de detectar las posiciones de los puntos clave, como una simple resta de imágenes.

- Además de esto, la función de Diferencia de Gaussianas, proporciona una aproximación cerrada de la Laplaciana normalizada en escala de la Gaussiana $\sigma^2 \nabla^2 G$, tal y como estudió Lindeberg en 1994 [15]. Lindeberg demostró que la normalización de la Laplaciana mediante el factor σ^2 es necesaria para obtener una escala invariante.

En comparaciones experimentales muy detalladas, Mikolajczyk en 2002 [18] encontró que el máximo y el mínimo de $\sigma^2 \nabla^2 G$ producen las características más estables de la imagen, comparándolas con un rango de posibles funciones de la imagen como el Gradiente, el Hessiano, o la función de las esquinas de Harris.

La relación entre D(x,y,σ) y $\sigma^2 \nabla^2 G$, puede entenderse mediante la ecuación de difusión por calor (parametrizada en términos de σ, en lugar de $t = \sigma^2$).

$$\frac{\partial G}{\partial \sigma} = \sigma \nabla^2 G \quad (4)$$

Por esto, podemos observar que $\nabla^2 G$ puede ser calculada por la aproximación en diferencia finita de $\partial G / \partial \sigma$ usando la diferencia de escalas cercanas de kσ y σ:

$$\sigma\nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma} \tag{5}$$

y por lo tanto:

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2\nabla^2 G \tag{6}$$

Con esto, queda demostrado que cuando la función de Diferencia de Gaussianas tiene diferentes escalas por un factor constante, ya incorpora la normalización de la escala σ^2, necesaria para la Laplaciana invariante en escala.

El factor (k-1) de las ecuaciones, es una constante de todas las escalas y por lo tanto no tiene influencia en el proceso de la localización extrema.

El error de aproximación será cero cuando $k \approx 1$, pero en la práctica, nos encontraremos con que la aproximación no tiene casi impacto en la estabilidad de la detección de extremos o en la localización para diferencias significativas en la escala, tal como $k \approx 2^{1/2}$.

Una buena forma de construir D(x,y,σ), es convolucionar la imagen inicial con Gaussianas para producir distintas imágenes separadas por un factor constante de escala y espacio, k; como se muestra en la figura 9.

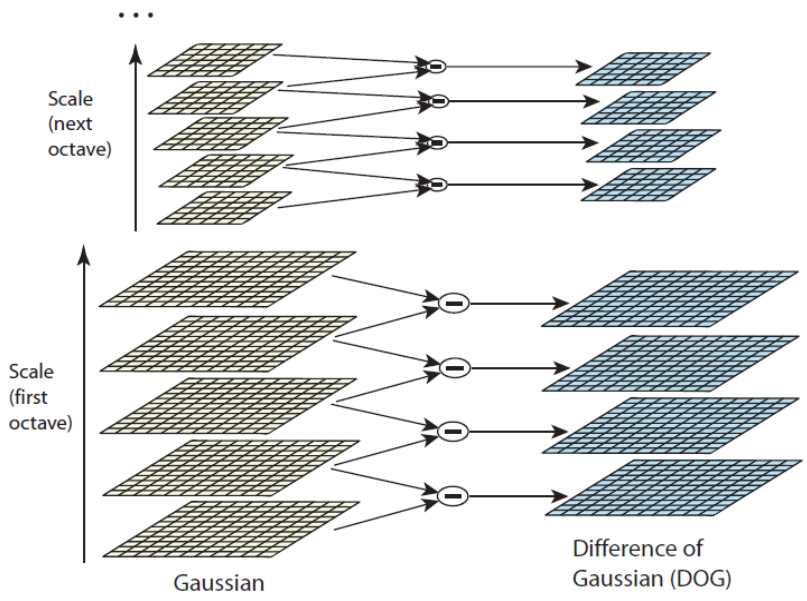


Imagen 9 - Formación de la Diferencia de Gaussianas (DOG).

Posteriormente se dividirá cada octava del "Escala-Espacio" en un número entero de intervalos, s, tal que $k = 2^{1/s}$ .Se deberán producir s+3 imágenes borrosas para cada octava. Las imágenes de escalas adyacentes serán restadas para producir la Diferencia de Gaussianas necesaria.

Una vez que la octava completa ha sido procesada se volverá a muestrear la imagen Gaussiana que tiene el doble del valor inicial de σ que tomen el segundo píxel de cada columna y fila. La precisión de las muestras relativas de σ no es diferente de las del comienzo de la octava previa, mientras que la cantidad de cálculos computacionales se reducen bastante de esta forma.

3.1.1.- DETECCIÓN DE EXTREMOS LOCALES

Para detectar los máximos y mínimos locales de D(x,y,σ), cada punto se compara con sus ocho vecinos de la imagen local y los nueve de (mismo punto y ocho vecinos) de las escalas anterior y posterior. Esto hace que se comparen con un total de 26 píxeles. Se seleccionará este punto siempre que sea menor o mayor que todos sus vecinos. Este proceso requiere poco coste computacional, ya que la mayor parte de los puntos que se deben controlar serán eliminados en los primeros controles. Se debe también tener muy en cuenta la frecuencia de muestreo en la imagen y la escala, necesaria para detectar de forma fiable los picos (máximos y mínimos).

Por desgracia, no existen unos valores de muestreo definidos con los que se puedan detectar todos los picos de los valores en las imágenes, al igual que estos extremos pueden o no ser cercanos entre sí. Esto se demuestra considerando un círculo blanco en un fondo negro, el cual tendrá un único máximo en todo el espacio y en escala de la imagen, donde el centro circular positivo de la función de Diferencia de Gaussianas coincidirá con el tamaño y la ubicación del círculo blanco en la imagen. Para una elipse muy alargada, habrá dos máximos cerca de cada uno de los extremos de la elipse.

Como la ubicación de los máximos es una función continua de la imagen, para algunas elipses con un anchura intermedia, se dará una transición de uno a dos máximos, arbitrariamente cercanos el uno del otro y muy cercanos a la transición. Por lo tanto, sólo podremos obtener una solución de compromiso entre la eficiencia y la integridad.

Como se demostró mediante varios experimentos, extremos muy cercanos son muy inestables cuando existen pequeños cambios en la imagen. Por eso se estudiaron una serie de frecuencias de muestreo que proporcionan unos resultados más realistas ante distintas situaciones.

3.1.2.- FRECUENCIA DE MUESTREO EN LA ESCALA

Se pretendió establecer mediante una serie de experimentos, una frecuencia de muestreo que maximizara la estabilidad de los extremos.

Para llevar a cabo esta labor, se partió de una tarea de emparejamiento de 32 imágenes reales de distintas características y procedencia (caras humanas, imágenes de paisajes, aéreas, etc). Cada una de las imágenes fueron sometidas a distintas transformaciones: rotaciones, incrustaciones de otros elementos, estiramientos, cambios de brillo y contraste o adicciones de ruido en la imagen.

Como estos cambios fueron sintéticos, fue posible la predicción exacta de la posición de las características de la imagen original que debían aparecer en la imagen transformada, permitiendo realizar una medición de la repetición y la precisión correctas para cada característica. Tras varias pruebas, se observaron los efectos obtenidos al variar el número de escalas por octava.

En el caso de muestrear cada imagen después de un cambio de rotación y escala elegidos al azar (entre 0,2 y 0,9 veces el valor original), y una adición de ruido (en un intervalo uniforme entre [-0,01, 0,01], donde los píxeles toman valores entre 0 y 1, equivalente a dar un valor de 6 bits por píxel), se dedujo que el número óptimo de escalas por octava era tres; por tanto ese es el número que se utilizó finalmente.

Se observó también que las repeticiones no mejoran a medida que se añaden más escalas por octava, ya que en estos casos el número de extremos locales aumenta, pero no siguen siendo estables, lo que causa que no se detecten tan bien en la imagen transformada.

De todas formas, a medida que aumentan las escalas, también lo hace el número de keypoints y el de "emparejamiento" de éstos entre las imágenes de test y las almacenadas; es por esto, por lo que dependiendo de la aplicación del programa, se use un mayor o menor número de escalas por octava, aunque a medida que usemos más, se aumenta el coste computacional y esto no interesa.

En definitiva, se observa que la función de "Escala-Espacio" realizada mediante la Diferencia de Gaussianas, da lugar a un gran número de keypoints, pero que es muy costoso computacionalmente detectarlos todos; por eso se usa un subconjunto más estable de puntos clave.

3.1.3.- FRECUENCIA DE MUESTREO ESPACIAL

Tal y como se determinó el valor de la frecuencia de muestreo de las escalas por octava, se debe dar un valor para la frecuencia de muestreo espacial. Teniendo en cuenta que los extremos espaciales pueden estar (o no) cerca, habrá una relación similar entre la frecuencia de muestreo y la tasa de detección.

Tras varios ensayos, se determinó que σ (el valor antes del suavizado) que se aplica a cada nivel de la imagen antes de construir el espacio de representación para una escala, continúa aumentando a medida que aumenta el número de repeticiones; sin embargo, también aumenta el coste (en términos de eficiencia), por lo que se eligió un valor óptimo de σ igual a 1,6.

Si se "presuaviza" la imagen antes de la detección de extremos, se deberán descartar las frecuencias espaciales más altas. Por lo tanto, para hacer un uso mayor de la entrada, la imagen se amplía para crear más puntos de la muestra que hayan estado presentes en la original. Se dobla el tamaño de la imagen de entrada mediante una extrapolación lineal antes de construir el primer nivel de la pirámide.

La operación de ampliación podría llevarse a cabo mediante el uso de filtros de compensación de subpíxeles de la imagen original, por lo que la duplicación de la imagen original da lugar a una aplicación más eficiente del algoritmo de búsqueda de máximos. Se asume que la imagen original tiene un desenfoque de al menos 0,5 (el mínimo necesario para evitar el efecto de solapamiento), la imagen tendrá el doble de espacio en su espacio entre píxeles. Esto implica que sea necesaria la realización de un pequeño suavizado antes de la primera octava del "Escala-Espacio".

El duplicado de la imagen, aumenta el número de puntos clave en un factor de cuatro veces los de la imagen original; pese a esto, no se mejoran los resultados en un factor tan elevado.

3.2.- LOCALIZACIÓN DE LOS KEYPOINTS

Tras encontrar un posible keypoint, comparando un píxel con sus vecinos, el siguiente paso es realizar un detallado de los alrededores de ese punto para observar la localización, escala y radio de las curvaturas mayores de dicho candidato. Esta información deja a los puntos preparados para ser rechazados si tienen un bajo contraste (son sensibles al ruido) o una mala localización (por ejemplo en un borde).

La implementación inicial de este enfoque tan simple de localización de los puntos clave, ideada por Lowe en 1999 [11], simplemente es capaz de encontrar el lugar y la escala del punto central de la muestra. Sin embargo, Brown (con ayuda de Lowe), desarrolló en el año 2002 [20], un método para ajustar una función cuadrática en tres dimensiones para los puntos locales de muestreo que determina la ubicación de los máximos.

Mediante diversos experimentos, se demostró que se mejora la estabilidad y localización de los puntos clave utilizando este método.

Este enfoque, usa el desarrollo de Taylor (por encima de términos cuadrados) de la función de "Escala-Espacio". Ésta, cambia de manera que el origen está en el punto de la muestra:

$$D(x) = D + \frac{\partial D^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D^T}{\partial x^2} x \tag{7}$$

Donde D y sus derivadas son evaluadas en la muestra y $x=(x,y,\sigma)^T$ es el desplazamiento de ese punto.

La localización del extremo $\hat{x}$, se determina cogiendo la derivada de la función con respecto a x y dando a este valor el valor Φ:

$$\hat{x} = \frac{\partial^2 D^{-1}}{\partial x^2} \frac{\partial D}{\partial x} \tag{8}$$

Como sugirió Brown, el Hessiano y la derivada de D, son aproximaciones usadas para la diferencia de vecinos de los puntos simples, y el sistema lineal (3x3) resultante, puede ser resuelto con un coste mínimo.

Si el desplazamiento $\hat{x}$ es mayor que 0,5 en cualquier dimensión, el extremo estaría más cerca de un punto de la muestra distinto. En este caso, el punto de muestra, se cambia y la interpolación se realiza sobre ese nuevo punto. El desplazamiento final $\hat{x}$, se añade a la localización del punto de la muestra que da una interpolación estimada para la localización del extremo.

La función evaluada en el extremo $D(\hat{x})$ es muy útil para rechazar extremos inestables y con un contraste bajo.

$$D(\hat{x}) = D + \frac{1}{2} \frac{\partial D^T}{\partial x} \hat{x} \tag{9}$$

Para los experimentos realizados por Lowe en su trabajo "Distinctive image features from Scale-Invariant Keypoints" de 2004 [29], todos los extremos con un valor de D($\hat{x}$) menor que 0,03 fueron descartados (ya que se asumieron valores de píxeles entre 0 y 1).

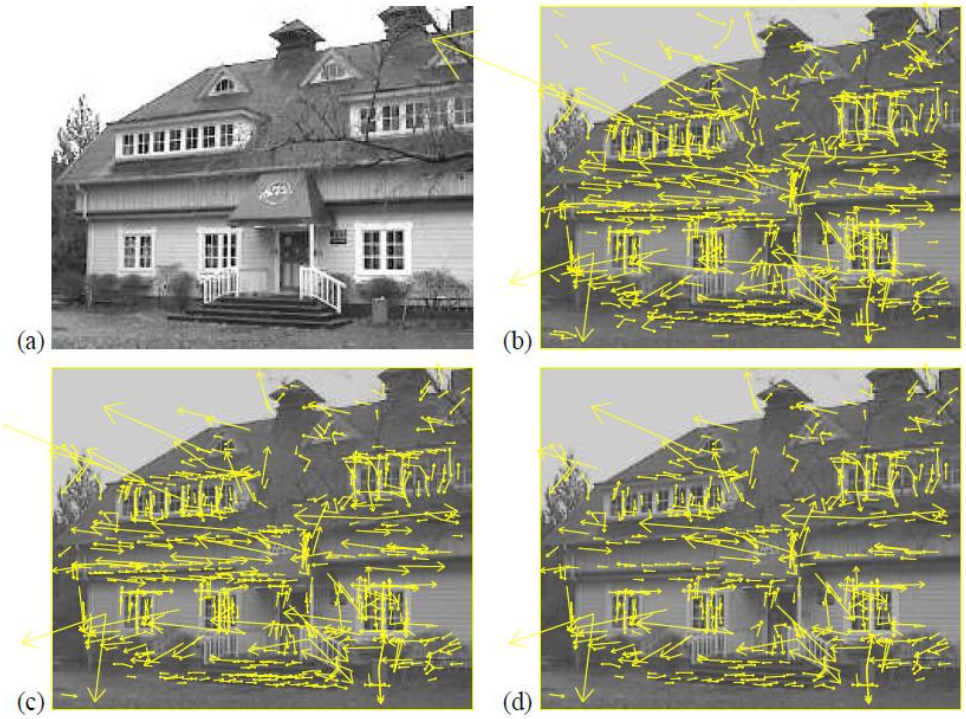


Imagen 10 - Etapas de elección de los puntos clave.

En la imagen 10, tomada de dicho experimento, se muestran los efectos de la selección de puntos clave en una imagen natural.

Con el fin de evitar un gran desorden, se tomó una imagen de baja resolución (233x189). Los keypoints, se exponen como vectores que muestran la ubicación, escala y orientación de sí mismos.

La figura muestra los distintos estados de la selección de puntos clave:

- Comenzando por arriba e izquierda, se puede observar la imagen original.
- Siguiendo hacia la derecha, se aprecian los 832 keypoints (máximos y mínimos) obtenidos mediante la función de Diferencia de Gaussianas.
- Abajo a la izquierda, podemos ver los 729 puntos clave resultantes de aplicar un umbral de mínimo contraste (D($\hat{x}$) $< 0{,}03$) a la imagen anterior.
- El último cuadro, muestra los 536 keypoints supervivientes tras la aplicación de un umbral de las curvaturas principales.

3.2.1.- ELIMINACIÓN DE LOS BORDES

Para obtener estabilidad, no es suficiente con rechazar los puntos clave con bajo contraste. La función de Diferencia de Gaussianas tendrá una respuesta firme a lo largo de los bordes,

aunque la localización en estas zonas no está bien determinada y por lo tanto es bastante inestable para pequeñas cantidades de ruido. Un pico mal definido en la función de Diferencia de Gaussianas tendrá una gran curvatura a través de los bordes, pero pequeña en la dirección perpendicular a los mismos.

Las curvaturas principales pueden ser calculadas mediante una matriz Hessiana de 2x2, a la que se denominará H, calculada en la localización y la escala del punto clave.

$$H = \begin{pmatrix} D_{XX} & D_{YX} \\ D_{XY} & D_{YY} \end{pmatrix} \tag{10}$$

Las derivadas, son estimadas tomando las diferencias de puntos de muestreo vecinos. Los valores de H son proporcionados a la curvatura principal de D.

Si se observa el enfoque de Harris y Stephens de 1988 [5], se puede evitar el cálculo explícito de los valores de H, ya que sólo se mirará su relación.

Llamando α al valor de mayor longitud y β al de menor., se podrá calcular la suma de los valores propios de la traza de H y su producto del determinante.

$$Tr(H) = D_{XX} + D_{YY} = \alpha + \beta \tag{11}$$
$$Det(H) = D_{XX} \cdot D_{YY} - (D_{XY})^2 = \alpha \cdot \beta \tag{12}$$

En el caso de que el determinante sea negativo, las curvaturas tienen un signo distinto al de los puntos descartados por no ser extremos.

Si r es la relación entre las magnitudes mayor y menor de los valores propios, tal que α = rβ:

$$\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r} \tag{13}$$

Solo dependerá de la relación entre los valores propios en lugar de sus valores individuales.
La cantidad $(r+1)^2/r$ es un mínimo cuando los dos valores propios son iguales y aumentan con r.

Por lo tanto, para comprobar que la relación de las curvaturas principales está por debajo de un umbral r, sólo se tiene que comprobar si:

$$\frac{Tr(H)^2}{Det(H)} = \frac{(r+1)^2}{r} \tag{14}$$

Es muy eficiente de calcular, ya que son necesarias menos de 20 operaciones de punto flotante para comprobar cada punto clave.

3.3.- ASIGNACIÓN DE LA ORIENTACIÓN

Para asignar una orientación coherente a cada punto clave, se observan las propiedades locales de una imagen, la descripción de los keypoints se puede representar en relación con esta orientación y así conseguir invarianza respecto a la rotación de la imagen.

Este enfoque contrasta con la orientación invariante de los descriptores de Schmid y Mohr de 1997 [10], en el que cada descriptor de la imagen se basa en una medida invariante a la rotación. La desventaja principal, es que se limitan los descriptores que se pueden utilizar y se descarta información de la imagen al no exigir que todas las medidas se basen en rotaciones coherentes. Tras experimentar con varios valores para realizar una asignación local, se encontró una aproximación para obtener resultados más estables.

La escala de los puntos clave, se usa para seleccionar la imagen Gaussiana suavizada (L) con el valor mayor de escala, de modo que todos los cálculos se realicen en un comportamiento invariante de la escala.

Para cada muestra de la imagen *L(x,y)* y utilizando el valor de la escala en ese punto, se calcula previamente la magnitud del gradiente m(x,y) y su orientación *Θ(x,y)* mediante estas ecuaciones:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \quad (15)$$

$$\theta(x,y) = tan^{-1} \frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)} \quad (16)$$

El histograma de la orientación, se forma a partir de las orientaciones del gradiente de los puntos de muestreo que rodean el keypoint. Dicho histograma, tiene los 360 grados divididos en 36 contenedores de 10 grados cada uno. Cada muestra añadida al histograma, se pondera por la magnitud del gradiente y por una ventana circular de ponderación Gaussiana con σ = 1,5 veces mayor que la escala del punto clave.

Los picos que aparecen en el histograma de la orientación se corresponden con las direcciones dominantes de los gradientes locales. El pico más alto se detecta en el histograma, junto con los picos locales (que sean al menos un 80% igual de altos que el pico más alto) para crear un punto clave en esa dirección.

Por lo tanto, para localizaciones con muchos picos de magnitud similar, habrá puntos clave creados en la misma posición pero con diferentes orientaciones. Solo al 15% de los puntos se les asignan distintas orientaciones, pero éstas contribuyen significativamente a la estabilidad de las posteriores búsquedas. Una parábola sería adecuada para los interpolar la posición del pico de mayor precisión mediante los tres valores del histograma más cercanos a cada uno de los picos.

Mediante la experimentación con las localizaciones, escala, orientación y asignaciones bajo distintas cantidades al azar de ruido, rotación y escala en imágenes, se pudo demostrar que las características SIFT son resistentes a grandes cantidades de ruido, y que las principales causas de errores son la ubicación y la detección de la escala.

3.4.- EL DESCRIPTOR LOCAL DE LA IMAGEN

Las operaciones previas, asignan una localización, escala y orientación en la imagen para cada uno de los keypoints. Estos parámetros imponen un sistema de coordenadas 2D para describir la región de la imagen local, y por lo tanto proporcionar invarianza a esas medidas.

Posteriormente, se calculará un descriptor de la región local de la imagen tan invariante a las variaciones de iluminación o rotaciones en 3 dimensiones como sea posible.
Una solución obvia sería mostrar la intensidad de la imagen local alrededor del punto clave con una escala apropiada usando una medida de la correlación normalizada. Sin embargo, la correlación simple de los pedazos de la imagen es muy sensible a los cambios que causan los registros erróneos de las muestras, tales como cambios de vista en 3D, o deformaciones no rígidas.

Edelman propuso un buen enfoque a este problema en 1997 [51], demostrando que si se basaba en un modelo de visión biológica, particularmente en las neuronas del complejo de la corteza visual primaria, éstas respondían al gradiente en una determinada orientación y frecuencia, pero la localización de dicho gradiente en la retina permitía pequeños cambios del campo receptivo en lugar de localizarse con precisión.

Supuso que la función de este complejo de neuronas, permitía el reconocimiento de objetos 3D en distintos puntos de vista. Se realizaron experimentos detallados usando modelos de objetos 3D y figuras de animales que mostraban la adecuación de los gradientes (mientras que permitían cambios en la posición), en los que se obtuvieron mejores resultados para las rotaciones en 3D. Por ejemplo, la correlación del 94% de los gradientes aumentaba un 35% la precisión del reconocimiento cuando se usaban objetos girados 20 grados, usando un modelo de complejo celular.

3.4.1.- REPRESENTACIÓN DEL DESCRIPTOR

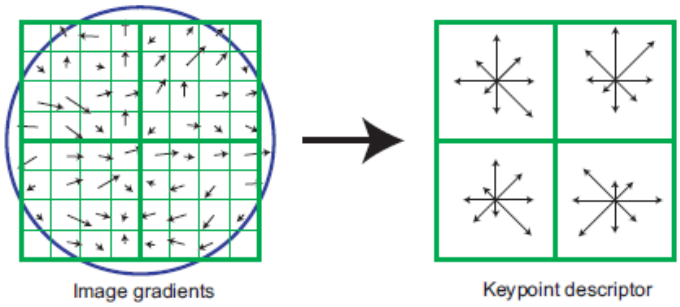


Imagen 11 - Creación del keypoint a partir de los gradientes.

Como puede verse en la imagen 11, se calculan en primer lugar las magnitudes del gradiente y las orientaciones de la imagen alrededor del punto clave, usando la escala de dicho punto para seleccionar el nivel de desenfoque Gaussiano de la imagen.

Para lograr la invarianza en la orientación, las coordenadas del descriptor y las orientaciones del gradiente se rotan en relación con la orientación del keypoint. Para obtener una mayor eficiencia, los gradientes son precalculados para todos los niveles de la pirámide, como ya se describió anteriormente. En la figura, estos gradientes se muestran como flechas en la parte izquierda de la imagen.

Una función de ponderación de Gauss con σ igual a la mitad del ancho de banda de la ventana del descriptor se usa como signo del peso de la magnitud de cada punto de muestreo; esto se ilustra con una ventana circular en el lado izquierdo de la figura. Los puntos centrales tendrán más peso que los más alejados del centro. El propósito de la ventana de Gauss es evitar los cambios bruscos en el descriptor con pequeños cambios en la posición de la ventana, y dar

menos énfasis a los gradientes más alejados del centro, pues éstos están más afectados por errores de registro erróneos.

Los descriptores de los keypoints aparecen en la parte derecha de la imagen. Se permiten cambios en la posición del gradiente mediante la creación de un histograma de orientación de regiones de 4x4 muestras. La figura muestra 8 direcciones para cada histograma de orientación, donde las longitudes de las flechas corresponden a las magnitudes de las entradas del histograma en esa dirección.

Una muestra de gradiente a la izquierda de la imagen puede cambiar hasta cuatro posiciones de la muestra mientras siga contribuyendo al histograma de la derecha, logrando así el objetivo de permitir mayores cambios en la posición local. Es importante evitar todas las fronteras que afecten a cambios muy bruscos del descriptor, como los cambios de suavidad de las muestras, que afectarían de un histograma al otro o de una orientación a otra. Por lo tanto, la interpolación trilinear se usa para distribuir el valor de cada muestra del gradiente en los compartimentos adyacentes del histograma.

En otras palabras, cada entrada en cada una de las regiones se multiplica por una constante (1-d) para cada dimensión, donde d es la distancia de la muestra al valor central del compartimento en unidades del espacio del histograma del contenedor.

El descriptor se forma a partir de un vector que contiene los valores de todas las orientaciones de las entradas del histograma correspondientes a las longitudes de las flechas del lado derecho de la imagen. La figura muestra un array 2x2 de histogramas de orientación, pero diversas pruebas demostraron que los mejores resultados se logran con arrays de 4x4 histogramas con 8 orientaciones cada uno, por lo tanto 128 elementos para cada keypoint. Por último, el vector de características se modifica para reducir los efectos de los cambios de iluminación.

En primer lugar, el vector es normalizado por una unidad de longitud. Un cambio en el contraste de la imagen en la que cada valor de píxel se multiplica por una constante, multiplicará los gradientes por la misma constante, por lo que este cambio de contraste será cancelado por este vector de normalización. Un cambio en el brillo en el que una constante se agrega a cada píxel, no afectará a los valores de los gradientes, ya que se calcula a partir de las diferencias de píxeles.

Por lo tanto, el descriptor es invariante a cambios de iluminación; sin embargo los cambios no lineales de iluminación, que pueden ocurrir por la saturación de la cámara y afectan a las superficies 3D con distintas orientaciones y en distintas cantidades, pueden causar un gran cambio en las magnitudes de algunos gradientes, pero tienen menos posibilidades de afectar a las orientaciones de los mismos.

De modo que se reducirá la influencia de grandes magnitudes del gradiente mediante una umbralización de los valores en el vector de características que sean menores que 0,2 y después se renormalizará la unidad de longitud.

Esto significa que la adecuación de las magnitudes de los gradientes grandes no es tan importante, y que la distribución de las orientaciones de los mismos es muy importante. El valor del umbral (0,2) se determinó experimentalmente usando imágenes que contenían distintas iluminaciones para los mismos objetos en 3D.

3.4.2.- PRUEBAS PARA LOS DESCRIPTORES

Existen dos parámetros que se podrán usar para variar la complejidad de los descriptores: el número de orientaciones en los histogramas (r) y la anchura del array de histogramas de orientación (n · n). El tamaño del vector del descriptor resultante será de rn^2.

A medida que la complejidad del descriptor aumenta, será capaz de discriminar mejor en grandes bases de datos, aunque también será más sensible a la forma de las distorsiones y oclusiones.

Para las distintas pruebas realizadas, se utilizaron transformaciones de superficies planas de hasta 50 grados y con un 4% de ruido añadido, sobre una base de datos de 40.000 keypoints.

Con diversos experimentos usando estos elementos y donde se variaron el número de orientaciones y el tamaño de los descriptores, se demostró que con un valor de n=1, la discriminación es demasiado pobre, pero los resultados mejoran hasta un valor de vector 4x4 con 8 orientaciones por histograma.

A partir de esos valores, el añadido de más orientaciones puede provocar que el descriptor sea más sensible a la distorsión. Estos resultados, se asemejan a los encontrados al cambiar el punto de vista o el ruido, aunque en algunos casos de discriminación sencillos, los resultados mejoran con arrays de 5x5 y mayores tamaños de los descriptores. Finalmente, se escogen unos valores de n=4 y r=8, dando como resultado vectores de 128 dimensiones.

Este número de dimensiones puede parecer alto, pero se comprobó que estos valores funcionan mejor que otros (menores que ellos) en tareas de compensación, y que el coste computacional continúa siendo bajo cuando se usan métodos de aproximación del vecino más próximo que se describirán a continuación.

3.4.3.- SENSIBILIDAD A CAMBIOS AFINES

Para comprobar cuál es el mejor método para detectar los cambios afines, se examinaron distintas situaciones. Estos experimentos observaron la fiabilidad de la localización de los puntos clave mediante la detección de la escala, la asignación de la orientación y la búsqueda mediante el vecino más cercano en una base de datos en la que variaba la rotación (punto de vista de una superficie plana).

A medida que se aumenta la distorsión afín, se reduce la fiabilidad, pero la precisión de aciertos se mantiene por encima del 50% con cambios de rotación de 50 grados. Para lograr coincidencias entre keypoints fiables en ángulos de visión amplios, los detectores invariantes podrían realizarse seccionando la imagen en regiones y observando uno a uno dichos pedazos. Ya se demostró anteriormente que este método no es del todo invariante, ya que las secciones en las que se debe actuar no son totalmente invariantes.

El método que mejores resultados parece dar, es el propuesto por Mikolajcz en 2002 [18], que realiza experimentos detallados con los detectores afines de Harris. En este caso, se descubrió que la fiabilidad es inferior con menores cambios de vista (rotaciones), pero que se mantiene en un valor cercano al 40% en cambios de ángulo de 70 grados, lo que proporciona mejores resultados en cambios afines extremos. Como desventajas principales, se encuentra el mayor

valor de cómputo, una reducción del número de puntos clave detectados y un menor grado de estabilidad con pequeños cambios afines bajo ruido.

En el entorno práctico, los giros permitidos en objetos 3D son menores que en las superficies planas, así que la invarianza afín no suele ser el factor que limita la capacidad de búsqueda en cambios de puntos de vista. Si se necesitara un gran valor de invarianza (por ejemplo para superficies no planas), es posible utilizar el enfoque de Pritchard y Heidrich de 2003, en el que las características adicionales SIFT se generan a partir de la cuarta versión de la transformada afín de la imagen de entrenamiento correspondiente a cambios de puntos de vista de 60 grados.

Esto permite el uso de características SIFT estándar sin un coste adicional añadido en lugar de cuando se realiza un procesamiento para reconocer una imagen, pero los resultados aumentan en un factor 3 el tamaño de la base de datos de características.

3.4.4.- RECONOCIMIENTO EN GRANDES BASES DE DATOS

La única cuestión importante que falta para medir la distinción de características es la forma en la que la fiabilidad de la búsqueda varía como una función del número de características de la base de datos emparejadas. La mayor parte de los ejemplos se generaron mediante una base de datos de 32 imágenes con unos 40.000 puntos clave cada una, pero en el caso de medir la capacidad de reconocimiento de grandes bases de datos, se usaron 112 imágenes con rotaciones de 30 grados y un 2% de ruido añadido, además de las habituales rotaciones y cambios de escala al azar.

Tras este sencillo experimento, se observó que la diferencia entre la búsqueda con el vecino más próximo y la búsqueda con la ubicación, escala y orientación correctas es mucho más pequeña, lo que indica que los errores entre ambos métodos se deben más bien a un problema en la localización, incluso con bases de datos muy grandes.

3.5.- APLICACIÓN AL RECONOCIMIENTO DE OBJETOS

Este tipo de transformada se usa sobre todo para el reconocimiento y la clasificación de objetos en desorden y oclusiones. Para el reconocimiento de objetos, se realiza en primer lugar una búsqueda de coincidencias entre los puntos clave extraídos los almacenados en la base de datos, procedentes de las imágenes de entrenamiento.

Muchas de estas coincidencias iniciales serán erróneas debido a las características que son muy comunes en varias imágenes, y a las que se confunden con el fondo. Por lo tanto, se usarán grupos de 3 características coincidentes sobre un mismo objeto y posteriormente serán observadas cada una de ellas de forma individual, realizando un ajuste geométrico de sus posiciones para el modelo que se esté usando aceptando o rechazando el grupo.

3.5.1.- BÚSQUEDA DE KEYPOINTS COINCIDENTES

La coincidencia más probable para cada punto clave se encuentra mediante el vecino más próximo en la base de datos de keypoints de las imágenes de entrenamiento. El vecino más cercano es el punto clave con mejor distancia euclídea al vector descriptor invariante.
Sin embargo, muchas características de una imagen no tendrán ningún resultado correcto en la base de datos de entrenamiento, ya que provienen de desórdenes o de fondos que no se detectaron en la fase de entrenamiento. Por lo tanto, es útil obtener una forma de deshacerse de este tipo de características no coincidentes con ninguna otra de la base de datos.

Un umbral global no da buenos resultados, pues algunos descriptores son más discriminantes que otros. Por esto, se buscará una medida más eficaz comparando la distancia del vecino más cercano con el segundo vecino más cercano; si hay varias imágenes de entrenamiento, se define como segundo vecino más cercano al vecino más cercano que proviene de un objeto distinto del que proviene el primer vecino. Esta medida se realiza de forma correcta, puesto que las coincidencias correctas necesitan tener un vecino más cercano significativamente mayor que el segundo vecino correcto para lograr coincidencias fiables poniendo un umbral acorde a ese vecino.

Para coincidencias incorrectas, habrá probablemente un número de coincidencias falsas con distancias similares debido a la gran dimensión del espacio de características almacenadas. Se podrá pensar en la segunda coincidencia más cercana como aquella que proporciona una estimación de la densidad de las coincidencias falsas dentro de esa porción del espacio de características, y al mismo tiempo identifica casos específicos de ambigüedad de características. Las funciones de densidad de coincidencias correctas e incorrectas muestran que las coincidencias que tenían un vecino más cercano como coincidencia correcta tienen una función de probabilidad centrada en una proporción mucho menor que las que el vecino más cercano era una coincidencia incorrecta.

Para uno de los ejemplos de aplicaciones para el reconocimiento de objetos se rechazaron coincidencias que tenían una relación de distancia superior al 0,8, lo que elimina el 90% de las coincidencias falsas y sólo un 5% de coincidencias correctas. Esta cifra fue generada a partir de imágenes con cambios de escala y orientación al azar, rotaciones de 30 grados y adicción de ruido de un 2% en bases de datos de 40.000 keypoints.

3.5.2.- INDEXACIÓN EFICIENTE DEL VECINO MÁS PRÓXIMO

No se conocen algoritmos que permitan identificar a los vecinos más cercanos de puntos en espacios de dimensión altos que sean más eficientes que la búsqueda exhaustiva. Los descriptores de keypoints tienen un vector de 128 dimensiones de características, y los mejores algoritmos, como el de árboles k-d de Friedman (1977) [52], no proporcionan búsquedas exhaustivas más rápidas para espacios dimensionales de más de diez dimensiones.

Por lo tanto, se usa un algoritmo de aproximación denominado BBF (Best Bin First), creado por Lowe y Beis en 1997 [53]. Es una aproximación, en el sentido en que devuelve el vecino más cercano con una probabilidad alta. Este algoritmo, utiliza una búsqueda modificada para el algoritmo de árbol k-d, tal que los contenedores en el espacio de características se buscan en el orden de la menor distancia de la ubicación que se esté tratando en cada momento.

Este orden de prioridad de la búsqueda se usó por primera vez en 1993 por Aria y Mount [54]; fue también por ellos, en 1998 [55] por lo que se obtuvieron estudios adicionales de las propiedades computacionales de este orden de prioridad de la búsqueda.

Una respuesta aproximada se puede devolver con un bajo coste, deteniendo la búsqueda después de que un número determinado de contenedores haya sido examinado.
En general, se corta la búsqueda cuando se han comprobado 200 candidatos al vecino más próximo. Para bases de datos de 100.000 keypoints, se acelera la búsqueda del vecino más próximo en dos órdenes de magnitud, y los resultados en menos de un 5% de reducción de los puntos coincidentes correctos.

Una de las razones por las que el algoritmo BBF funciona bien para este tipo de problemas, es que solo se tienen en cuenta las coincidencias en que los vecinos más cercanos están a menos de 0,8 veces del segundo vecino más cercano (como ya se ha descrito anteriormente), y por lo tanto, no existe necesidad de resolver los casos más difíciles en los que muchos vecinos se encuentran a distancias similares.

3.5.3.- AGRUPAMIENTO CON TRANSFORMADA DE HOUGH

Para maximizar el rendimiento del reconocimiento de objetos para objetos pequeños o muy ocluidos por otros objetos, se desea identificar los objetos con el menor número posible de características coincidentes. Se ha demostrado, que es posible realizar un reconocimiento fiable con tan sólo tres características.

Una imagen típica tiene unas 2000 (o más) características que pueden proceder de distintos objetos o del fondo. Mientras que el test del radio de la distancia descrito en la sección 3.5.1 permitía descartar muchas de las coincidencias erróneas provocadas por el fondo, no eliminaba las coincidencias con otros objetos válidos pero que no son correctos, y todavía a menudo es necesario identificar subconjuntos correctos de coincidencias que contienen menos del 1% de *INLIERS* entre el 99% de *OUTLIERS*.

Muchos métodos robustos conocidos como el RANSAC o el de menor mediana de cuadrados, proporcionan un bajo rendimiento cuando el porcentaje de *Inliers* está por debajo del 50%.
Afortunadamente, se puede obtener un mejor rendimiento con agrupamientos de características en representación el espacio usándola Transformada de Hough (Hough, 1962 [56]; Ballard, 1981 [57]; Grimson, 1990 [58]).

La Transformada de Hough identifica grupos de características con una interpretación coherente usando cada característica para considerar todas las representaciones de objetos que son coherentes con la característica. Cuando grupos de características se encuentran consideradas por la misma representación de un objeto, la probabilidad de que esa interpretación sea la correcta es mucho mayor que cuando solo una característica considera esa representación del objeto.

Cada uno de los puntos clave especificará cuatro parámetros: localización 2D, escala y orientación. Cada punto clave encontrado en la base de datos tiene un registro relativo a la formación de la imagen de entrenamiento en la que se encontró ese keypoint. Por lo tanto, es posible crear una entrada de la Transformada de Hough prediciendo el modelo de orientación, localización y escala de la supuesta coincidencia. Esta predicción tiene grandes errores en los límites, ya que la transformada similar implícita por esos cuatro parámetros es solo una

aproximación de los seis grados de libertad de la representación de objetos 3D, y no tiene en cuenta deformaciones no rígidas.

Por eso, se usarán tamaños de contenedor de 30 grados para la orientación, un factor de escala de 2 y 0,25 veces la máxima dimensión de la imagen de entrenamiento (usando una escala de predicción) para la localización.
Para evitar el problema de los efectos del límite en la asignación del contenedor, cada coincidencia con un punto clave considera los dos contenedores más cercanos en cada dimensión, dando un total de 16 entradas para cada hipótesis y un nuevo rango de representación ensanchado.

En la mayor parte de las implementaciones de la Transformada de Hough, se usa un array multidimensional para representar los contenedores. Sin embargo, muchos de estos contenedores permanecen vacíos, y es difícil de calcular el rango de posibles valores del contenedor debido a su dependencia mutua (por ejemplo, la dependencia de la discretización de la localización en la escala seleccionada). Estos problemas pueden evitarse usando una función Hash pseudoaleatoria de los valores del contenedor considerados para insertar en una tabla Hash unidimensional en las que las colisiones son detectadas fácilmente.

3.5.4.- SOLUCIÓN PARA LOS PARÁMETROS AFINES

La transformada de Hough se usa para identificar todos los grupos de más de tres entradas por cada contenedor. Cada grupo, está sujeto a un proceso de verificación geométrica en la que se lleva a cabo una solución de mínimos cuadrados para los mejores parámetros de proyección afín que relacionen la imagen de entrenamiento con la nueva imagen. Las transformaciones afines en rotaciones 3D de superficies planas bajo proyección ortogonal causan buenos efectos, pero la aproximación puede ser mala para rotaciones de objetos 3D no planos.

Una solución más general sería resolver la matriz fundamental (Luong y Faugeras, 1996 [59]; Hartley y Zisserman, 2000 [60]). Sin embargo, una solución a la matriz fundamental requiere un mínimo de 7 puntos para comparar solo 3 de la solución afín, y en la práctica requiere muchos más puntos coincidentes para mejorar la estabilidad. Se desearía realizar el reconocimiento con tan solo 3 puntos coincidentes, así la solución afín proporcionará un mejor punto de partida y se podrán ver los errores en la aproximación permitiendo errores residuales mayores. Si imaginamos colocar una esfera alrededor de un objeto, una rotación de la esfera de 30 grados, no moverá ningún punto dentro de ella más de 0,25 veces el diámetro proyectado de la misma.

Un enfoque más general, es el propuesto por Brown y Lowe en 2002 [20], en el que basan la solución inicial en una transformada de similitud, la cual, posteriormente tiende hacia la matriz fundamental en casos en los que se cuente con un número suficiente de puntos.

La transformación afín de un modelo de puntos $[x\ y]^T$ a un punto de la imagen $[u\ v]^T$ puede ser descrita como:

$$\begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{18}$$

donde $[t_x\ t_y]^T$ es el modelo de transformación y la rotación, escala y elasticidad afines están representados por los parámetros m_i.

Se desea resolver los parámetros de transformación, por lo que la ecuación anterior se reescribirá para reunir las incógnitas en un vector columna.

$$\begin{bmatrix} x & y & 0 & 0 & 1 & 0 \\ 0 & 0 & x & y & 0 & 1 \\ & & & \dots & & \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_x \\ t_y \end{bmatrix} = \begin{bmatrix} u \\ v \\ \vdots \end{bmatrix} \quad (19)$$

Esta ecuación muestra un solo punto, pero cualquier número de puntos adicionales puede ser aumentado, añadiendo por cada uno de ellos dos filas en la primera y la última matriz.
Se necesitarán, al menos tres puntos para proporcionar una solución.

Se puede escribir este sistema lineal como Ax=b. La solución de mínimos cuadrados para el parámetro x puede determinarse resolviendo las ecuaciones normales correspondientes:

$$x = [A^T A]^{-1} A^T b \quad (20)$$

que minimiza la suma de los cuadrados de las distancias que provienen de la proyección del modelo de localizaciones de las correspondientes localizaciones en la imagen.

Este modelo de mínimos cuadrados puede extenderse fácilmente a la solución de representación 3D y los parámetros internos de objetos articulados y flexibles (Lowe, 1991) [61]. Los *Outliers* pueden ahora ser eliminados para comprobar acuerdos entre cada característica de la imagen y el modelo.

Teniendo en cuenta la solución de mínimos cuadrados más exacta requeriremos ahora cada punto para estar de acuerdo dentro de la mitad del rango de error usado para los parámetros de los contenedores de la Transformada de Hough. Si menos de tres puntos siguen siendo puntos extremos después de descartar, ese contenedor se rechaza.

Como los valores extremos se descartan, la solución de mínimos cuadrados se vuelve a realizar con los puntos restantes y el proceso se repite. Además, se usa una fase de proceso del tipo “Top-Down” para añadir los puntos que coinciden con la proyección del modelo de posición. Estos pueden haberse perdido por la Transformada de Hough debido a la aproximación de la Transformada de Similitud de otros errores.

La decisión final de aceptar o rechazar una hipótesis del modelo está basada en un modelo probabilístico detallado en una investigación de Lowe en 2001 [28]. Este método calcula el número esperado de respuestas falsas del modelo planteado, dado el tamaño previsto para el modelo, el número de funciones en la región y la precisión del ajuste.

Posteriormente se realiza un análisis Bayesiano de la probabilidad de que el objeto esté basado en el número actual de características coincidentes encontradas. Se aceptará el modelo si la probabilidad final para una interpretación correcta es mayor a 0,98.

Para objetos que se proyectan a las regiones pequeñas de una imagen, tres características pueden ser suficientes para un reconocimiento fiable. Para objetos de gran tamaño que cubran la mayor parte de una imagen con textura, el número esperado de puntos fiables es mayor, y necesitaremos unos diez puntos de características.

3.6.- EJEMPLOS DE RECONOCIMIENTO REALIZADOS

Se han realizado varios trabajos de reconocimiento de objetos, aunque los más destacados son los del autor del algoritmo SIFT David Lowe. El aquí expuesto está tomado íntegramente de su trabajo de investigación "Distinctive Image Features From Scale-Invariant Keypoints" de 2004 [29].

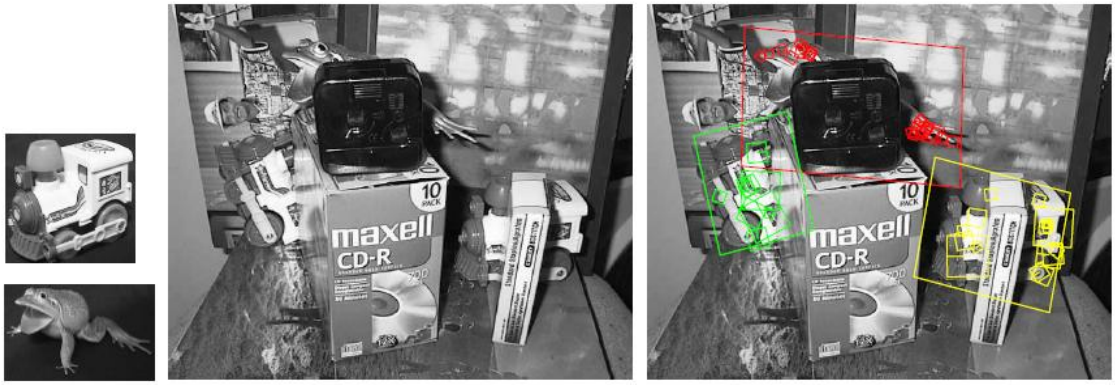


Imagen 12 - Ejemplo de reconocimiento de dos imágenes.

La figura muestra un ejemplo de reconocimiento de objetos de una imagen desordenada y con oclusiones, que contiene objetos 3D.

Las imágenes de entrenamiento del sistema (tren de juguete y rana) se muestran en la parte izquierda. La imagen central (de 600x480 píxeles) contiene estos objetos ocultos detrás de los demás y con un gran desorden, por lo que su detección no es inmediata, incluso para una persona. La imagen de la derecha muestra la correcta identificación.

Los puntos clave usados para el reconocimiento se muestran como cuadrados con una línea adicional para identificar la orientación. Los tamaños de los cuadrados, corresponden a las regiones de la imagen usadas para construir el descriptor. Se dibuja también un paralelogramo exterior alrededor de cada objeto reconocido, con los lados en los límites de las imágenes proyectadas en la formación de la transformación afín determinada durante el reconocimiento.

Otro de los experimentos que puede realizarse es el de la detección de objetos en movimiento, en las que la localización de un dispositivo móvil podía determinarse de forma correcta. En el mismo trabajo de investigación de Lowe, se realizó un experimento de estas características.

Todos los pasos de los experimentos realizados en esta investigación, se llevaron a cabo mediante implementaciones eficientes, en los que las tareas de reconocimiento se realizaron en menos de 0,3 segundos, con un Pentium 4 a 2 GHz. También se probaron los algoritmos en ordenadores portátiles con cámaras de vídeo conectadas y sobre una gran gama de condiciones.

En general, las superficies de textura plana se detectaron de forma fiable con rotaciones en profundidad de hasta 50 grados y bajo condiciones de luz que no proporcionaran brillo ni defectos de luz excesivos.

Para objetos 3D, la gama de rotaciones en profundidad para un reconocimiento fiable se reduce a 30 grados y los cambios de iluminación son mucho más perjudiciales. Por eso, para el reconocimiento de objetos 3D se realiza mejor mediante la integración de características de múltiples puntos de vista. Estos puntos clave se han aplicado también a problemas de localización de robots y cartografías.

3.7.- CONCLUSIONES DE LAS INVESTIGACIONES SIFT

Los keypoints SIFT son particularmente útiles, debido a su carácter distintivo, que permite la correcta búsqueda de un punto clave seleccionado en una gran base de datos de keypoints. Esta distinción se lleva a cabo mediante la creación de un vector de grandes dimensiones que representa los gradientes de la imagen en una región local de la misma. Los puntos clave demuestran ser invariantes a la rotación y a la escala de la imagen y robustos ante un gran margen de distorsiones, además de añadidos de ruido y cambios de iluminación.

Un gran número de puntos clave se extraen de imágenes típicas, que conducen a un reconocimiento fiable de pequeños objetos entre desórdenes. El hecho de que los puntos clave se detectan en todo el rango completo de escalas, significa que las pequeñas características locales están disponibles para buscar objetos grandes y pequeños ocluidos, mientras que los keypoints grandes sirven para imágenes sometidas a ruidos y desenfoques.
Su cálculo es eficiente, por lo que miles de puntos clave pueden extraerse de una imagen típica en tiempo casi real en ordenadores personales estándar.

Para el uso de puntos clave para reconocimiento de objetos mediante la transformada SIFT se utiliza la aproximación de la búsqueda del vecino más próximo, una transformada de Hough para identificar los grupos correctos del objeto planteado, y los mínimos cuadrados para representar la determinación y verificación final.

Las investigaciones llevadas a cabo para la optimización de los parámetros de la transformada SIFT sólo tratan imágenes en blanco y negro, aunque sería posible usar descriptores de colores mediante ING invariante a la iluminación (descrito por Funt y Finlayson en 1995 [62], y Brown y Lowe en 2002 [20]).

4.- DISEÑO E IMPLEMENTACIÓN

Tras sentar las bases y la posición del algoritmo SIFT en el mercado actual, se procede a diseñar un sistema que permita realizar la actividad para la que se ha desarrollado este proyecto: la investigación de los distintos métodos de reconocimiento de logotipos en imágenes.

En este apartado de la memoria, se comentará cómo se ha llevado a cabo el diseño de los distintos tipos de pruebas que el sistema será capaz de realizar y la implementación de los mismos.

4.1.- DISEÑO

4.1.1.- BLOQUES DEL SISTEMA

Se desea diseñar un sistema que permita realizar una serie de comprobaciones en la utilización de las distintas posibilidades de la Transformada SIFT para el reconocimiento de logotipos. Para ello, como ya se ha comentado anteriormente, se utilizará la herramienta VL_FEAT para Matlab.

En primer lugar será necesario diseñar los dos bloques más importantes del sistema, sobre las que se asentarán todas las investigaciones y utilidades que se realizarán a lo largo del proyecto. La primera de estas dos partes será el entrenamiento del sistema, y se encargará de extraer las características de los logotipos que más tarde se tratarán de identificar. La segunda será la encomendada a la tarea del reconocimiento de los logotipos que aparezcan en las imágenes que el usuario introduzca para estudiar el comportamiento del sistema.

Para obtener un uso dinámico de las características almacenadas, los resultados obtenidos durante el reconocimiento, el nombre y la ruta de la imagen que se estén tratando, etc., se creará un archivo de características accesible desde cualquier parte del sistema y denominado *features.mat,* que será accesible de un modo sencillo en todo momento. Dentro de este archivo se tendrán dos variables que indicarán la ruta y el nombre de la imagen que se esté tratando en cada momento (*pathName* y *fileName*), un vector de las características de las imágenes almacenadas durante el entrenamiento (*storedImages*) y otro para los resultados obtenidos en la búsqueda de coincidencias (*results*).

El diagrama de bloques en el que se basará la realización del sistema será el descrito en la figura 13, de la siguiente página. En ella se pueden observar los dos bloques principales del programa y cómo se relacionan con la interfaz de usuario.

Para la realización del entrenamiento y el test del programa se realiza un preprocesado de la imagen, en el que se realizarán una serie de acciones:

- Se observará si la imagen introducida en el sistema es soportada por el comando `imread` de Matlab. Los tipos de imágenes soportadas se almacenarán en el sistema mediante un vector de tipo *string* y serán: *bmp, gif, hdf, jpeg, jpg, pcx, png, tiff, y xwd.* También se tendrá en cuenta que estas extensiones podrán aparecer tanto en mayúsculas como en minúsculas.

- Se comprobará que las imágenes de entrada estén en escala de grises. Existen dos métodos para la conversión de las imágenes RGB a escala de grises[3].

- Se comprueba que los datos de las imágenes sean del tipo *single*.

- Se almacena el nombre de la imagen y su tamaño para un uso posterior.

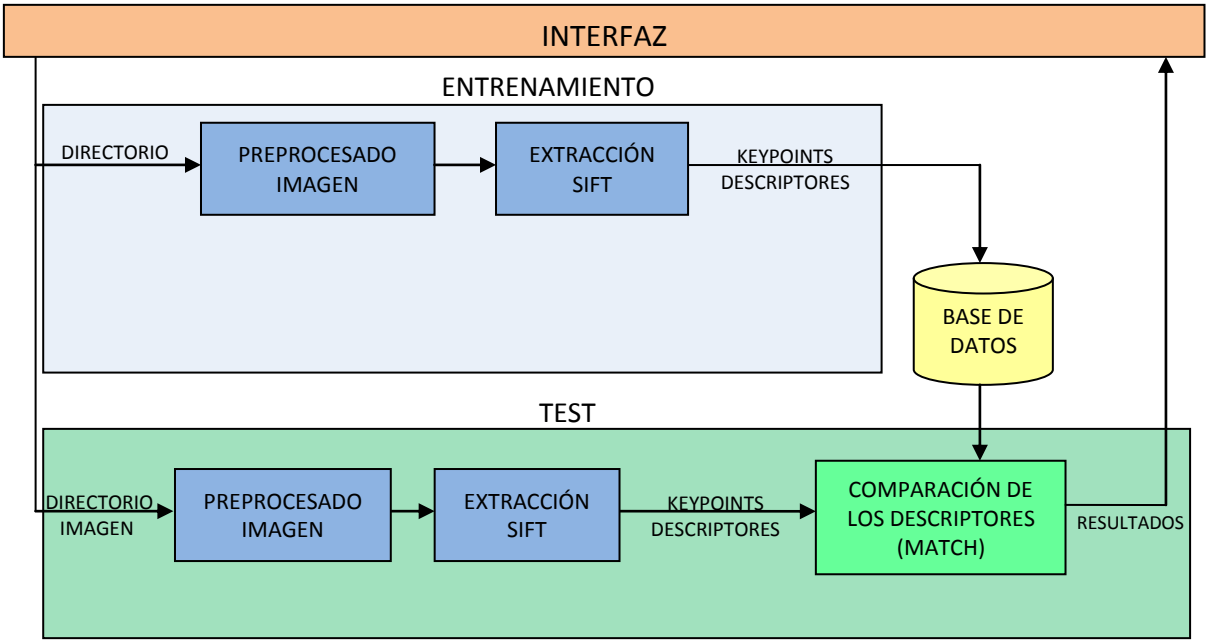
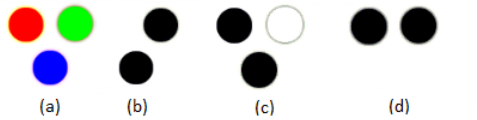


Imagen 13 - Diagrama de bloques del sistema.

[3] TRANSFORMAR IMÁGENES EN COLOR A IMÁGENES EN ESCALA DE GRISES [63]
Conociendo que una imagen en color está formada por tres capas monocromáticas (una para el color verde, otra para el rojo y otra para el azul) se puede recurrir a dos métodos para transformar dicha imagen a una escala de grises.

• El primero de los dos métodos consiste en elegir únicamente una de las tres capas monocromáticas en lugar de realizar cualquier operación. En este caso, se podría perder información de la imagen original, tal y como se puede observar en la siguiente imagen.



Capas de una imagen RGB; (a) Imagen en color; (b) Componente roja; (c) Componente verde; (d) Componente azul.

• El segundo método es mucho más efectivo. Consiste en obtener la luminancia mediante una combinación lineal de las tres capas de la imagen. Dicho parámetro se obtendrá mediante la siguiente expresión:

$$I(x,y) = 0{,}30 \cdot R(x,y) + 0{,}59 \cdot G(x,y) + 0{,}11 \cdot B(x,y) \tag{21}$$

Donde R es la componente roja del píxel con coordenada (x,y), G la componente verde, B la azul e Y la luminancia resultante en dicha coordenada.
Aplicando esta ecuación se puede observar que se obtiene una imagen en escala de grises en la que la componente de luminancia tiene la información necesaria para el procesado de la imagen y no se pierde información. Así pues este será el método utilizado por Matlab para realizar la conversión.

Imagen RGB y componente de luminancia.

El siguiente paso del diagrama para los bloques de entrenamiento y test será el encargado de la extracción de las características necesarias para el posterior reconocimiento. VL_FEAT, realiza una extracción de *keypoints* y *descriptores* para cada imagen de entrada mediante la realización de los métodos de la Transformada SIFT descrita en el *Apartado 3* de este proyecto, utilizando la sentencia `vl_sift.`

Los *keypoints* son los puntos de la imagen que la Transformada SIFT ha elegido para representarla. Como se describió anteriormente, son más que suficientes para describir de forma correcta la imagen y posteriormente poder reconocerla. Para cada *keypoint* se almacenan cuatro vectores que contendrán la posición del mismo (x e y), la escala y la orientación.

Los *descriptores* son vectores asociados a cada uno de los *keypoints*. Tienen una longitud de 128 elementos que detallan el punto clave para su posterior uso. Los valores de los descriptores están contenidos entre 0 y 255, aunque en este caso se normalizarán de 0 a 1 dividiendo cada término del vector entre 255.

Dos umbrales que se deben tener en cuenta durante la realización de la Transformada SIFT son los de pico y bordes (*peak* y *edge*). Éstos gestionan los valores que los picos del histograma y los bordes que en las imágenes se deben cumplir para considerar el punto estudiado como un posible *keypoint*.

- El valor *peak* (pico) deberá ser al menos el 80% de alto que el pico más alto y los picos locales del histograma de la imagen estudiada. El valor máximo para este umbral será 0.
- El valor *edge* (borde) controlará la forma en que los bordes de los objetos contenidos en las imágenes sean tratados. El valor máximo para este umbral será 10.

Los valores máximos para ambos umbrales son los que están predefinidos en la realización de la Transformada SIFT mediante `vl_sift`, por lo que darán el número máximo de puntos clave (*keypoints*) y *descriptores*. Por esto, el único motivo por el que estos umbrales deberán cambiarse será para conseguir un número menor de puntos clave. Para el caso que aborda este proyecto será una forma de realizar una restricción durante la extracción de características, por lo que también servirán como umbrales para la decisión de los valores del mejor extractor de características. A continuación se muestran dos ejemplos de búsqueda de características con distintos valores de *pico* y *borde*:

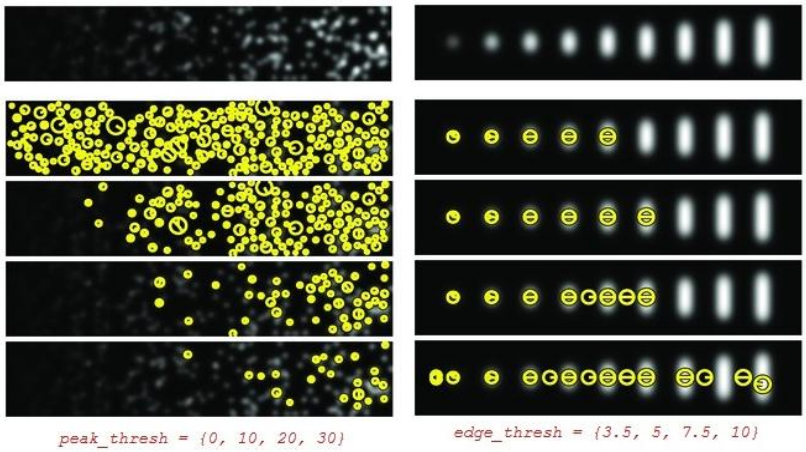


Imagen 14 - Características localizadas con distintos valores de pico y borde.

Durante el proceso de entrenamiento del sistema se recorrerá el directorio donde estén contenidas las imágenes y se almacenarán las características anteriormente descritas en la variable *storedImages* del archivo *features,* en la que cada una de las posiciones pertenecerá a una imagen de entrada y contendrá el nombre (*name*), tamaño (*sizeX* y *sizeY*), puntos clave (*keypoints*) y descriptores (*descriptors*). En cambio durante el reconocimiento no será necesario almacenar dichas características, pues la imagen se usará en el momento y los resultados serán guardados en otra variable distinta.

Cuando se realiza el test del sistema, se utilizará también un método de comparación (*Matching*) de los descriptores de las imágenes almacenadas y los extraídos en la imagen de entrada. El comando que se utilizará para realizar esta búsqueda de coincidencias entre los puntos clave será `vl_ubcmatch`, incluido dentro de las herramientas proporcionadas en el paquete de VL_FEAT. El método de búsqueda de coincidencias se basa en la distancia euclídea[4] para encontrar los parecidos entre ambos descriptores y devuelve dos vectores (*matches* y *scores*) que se almacenarán dentro de la variable *results* del archivo *features*.

El vector *matches* es un listado de dos columnas. En la primera, se indica el número correspondiente al keypoint de la imagen actual y en la segunda el número de keypoint de la imagen almacenada en el entrenamiento y con la que se esté realizando la búsqueda de coincidencias en este momento y en el que se haya encontrado un parecido razonable (una distancia euclídea que indique una posible coincidencia entre keypoints).

El vector *scores* es un listado que contiene el valor de la distancia euclídea de las coincidencias obtenidas entre los descriptores de las dos imágenes tratadas en cada momento (la de entrada y la almacenada en el entrenamiento).

Tendremos un vector de cada tipo para cada una de las imágenes del entrenamiento con las que comparemos la imagen de entrada. El resultado de la búsqueda de coincidencias puede apreciarse en el siguiente ejemplo:

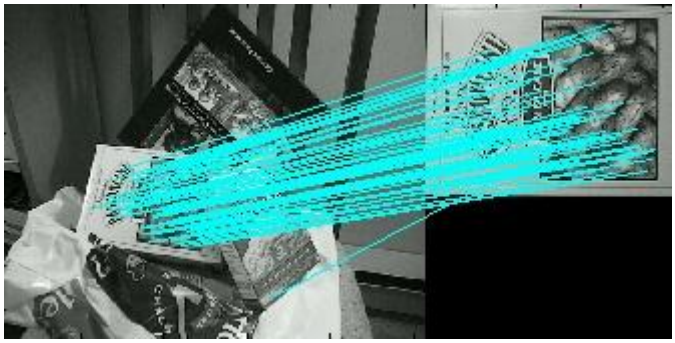


Imagen 15 - Ejemplo de búsqueda de coincidencias.

Tras la búsqueda de coincidencias y dentro del bloque de comparación de los descriptores, se realiza un filtrado de los resultados obtenidos, pues no todos serán válidos. Imponiendo un umbral, se recorre el vector de distancias euclídeas y se almacenan las que tengan un valor inferior a dicho umbral. Este valor, será uno de los que servirán para la realización de las pruebas del sistema y que llevará a unas conclusiones en la búsqueda de los mejores valores y métodos para el reconocimiento de logotipos en imágenes mediante la Transformada SIFT. Los

[4] La distancia euclídea es la distancia más corta medida entre dos puntos del espacio. En el caso general, esta distancia será la línea recta que una los dos puntos.

$$D(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (22)$$

Donde P_1 y P_2 son los puntos a tratar y (x_1,x_2), (y_1,y_2) son las posiciones de dichos puntos en el espacio.

scores y *matches* aceptados se almacenarán en sendas variables denominadas *scoresAccepted* y *matchesAccepted* en cada posición del vector *results* del archivo *features*. En el caso de utilizar el umbral impuesto por `vl_ubcmatch` los vectores *scoresAccepted* y *matchesAccepted*, serán los obtenidos al realizar la búsqueda de coincidencias directamente, pues este métodoserá el que se encargue de realizar el filtrado.

Siempre y cuando exista un número de coincidencias válidas superior a un porcentaje de puntos clave obtenidos durante la etapa de entrenamiento del sistema para cada una de las imágenes, se considerará que existe el logotipo buscado en el interior de la escena tratada. En este caso se buscarán y almacenarán las posiciones de los keypoints situados más a la derecha y abajo y más a la izquierda y arriba para su posterior marcado (*xMin, xMax, yMin* e *yMax*). Se almacenará también el nombre de la imagen del logotipo que aparece en la imagen tratada (*nameResult*) y se dará un valor positivo a una variable denominada *existLogo* en cada una de las posiciones del vector de resultados en la que se ha encontrado el logotipo.

En caso de que no exista ningún logotipo se almacenarán todas estas variables con valor nulo.

Para realizar las pruebas del sistema, se utilizará una base de datos de 100 logotipos distintos, con dos imágenes de cada uno de ellos para realizar el entrenamiento y una para el test del sistema, con lo que se deberá tener una base de datos de 300 imágenes distintas. En las imágenes utilizadas para realizar la comprobación y la investigación de los distintos métodos, se cambiará en un número determinado de ellas la rotación, escala, iluminación, introducción de oclusiones, introducción del logotipo en distintos escenarios, etc., para así poder observar los distintos casos de reconocimiento y poder concluir los resultados a los que el sistema llega.

Durante la realización de las pruebas, se comprobará el funcionamiento del sistema ante el reconocimiento de logotipos, ajustando todos los parámetros posibles y añadiendo alguno más para la realización de un etiquetado correcto.

4.1.2.- INTERFAZ DEL SISTEMA

Una vez pensados los dos bloques más importantes del sistema, se procede a la realización de la interfaz del programa. Será necesario que dicha interfaz cumpla todos los requisitos impuestos en el anterior apartado, puesto que la mayor parte de las actividades que el sistema podrá realizar estarán basadas en las dos partes más importantes del mismo (entrenamiento y test). Además se añadirán una serie de requisitos para que el uso de dicha interfaz sea mucho más dinámico:

- La interfaz debe ser sencilla y simple para que cualquier usuario pueda utilizarla.
- Se dará la posibilidad de reconocer logotipos individuales o colecciones de ellos dentro de un directorio.
- El sistema debe ser consciente en todo momento de las opciones que puede realizar y comunicárselas de alguna forma al usuario.
- Los ficheros implementados irán debidamente comentados (en inglés) para un una lectura y comprensión mayor por parte de los programadores que deban modificarlos en otro momento, etc.
- La opción de abandonar el sistema deberá darse únicamente en la pantalla principal del mismo, para evitar que el usuario pulse el botón de cerrar y deje abierto algún proceso.

- Se tendrá un archivo de características que servirá para intercomunicar los distintos procesos entre sí; en él se almacenarán las características realizadas durante el entrenamiento, la ruta y nombre de la imagen a reconocer y los resultados obtenidos en el reconocimiento. El nombre del archivo será *features.mat.*

A continuación se razona debidamente la cantidad de ficheros de Matlab necesarios para la realización de las actividades que el sistema debe tener. Se piensan en tres ventanas principales y otros cuatro procesos que no precisarán de ninguna interfaz gráfica, con lo que se obtienen siete archivos Matlab para un correcto funcionamiento e interconexión entre los distintos bloques del sistema.

El primero de los archivos a ejecutar se denominará *startup.m*, y será el encargado de la realización del inicio del programa, carga del paquete de herramientas VL_FEAT y la comprobación de si el archivo de características (*features*) ha sido anteriormente creado.

El siguiente paso será cargar en la pantalla la ventana del menú principal, incluida en otro archivo Matlab denominado *main.m*. Dicho menú se compondrá de un área reservado para mostrar el logotipo del sistema, otro para la comunicación con el usuario mediante mensajes de texto y cinco botones principales: uno para el entrenamiento del sistema ("*Train the system*"), dos para el reconocimiento tanto de imágenes ("*Recognize the logos of a image*") como de directorios ("*Recognize the logos of a directory*"), otro botón para los créditos del programa ("*About*") y el último para abandonarlo ("*Exit*"). Los botones de reconocer los logotipos estarán desactivados siempre y cuando no se haya realizado anteriormente un entrenamiento del sistema. La interfaz gráfica resultante para este menú será la que se muestra en la imagen 16.

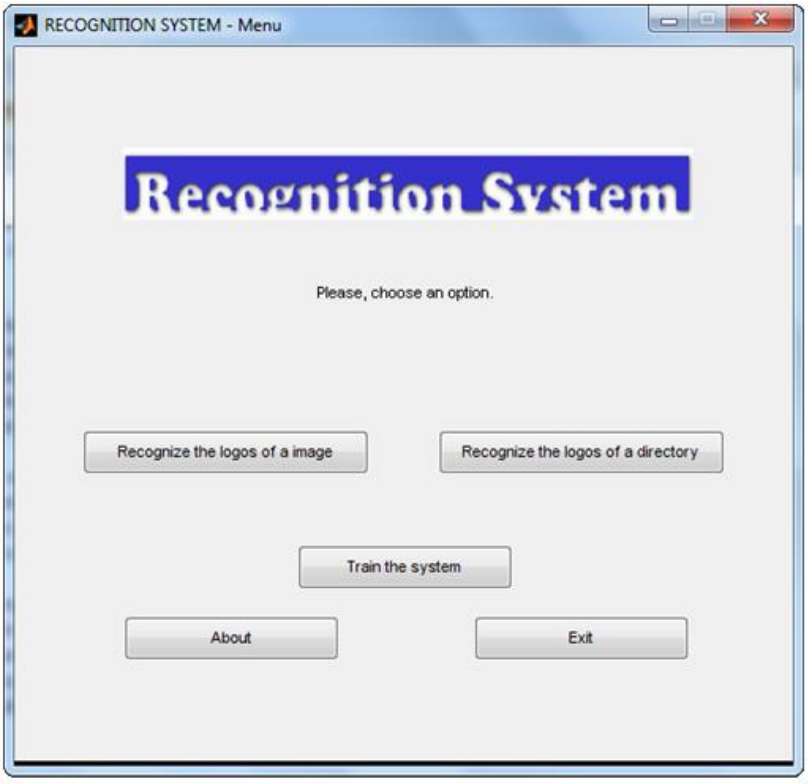


Imagen 16 - Ventana del menú principal.

El siguiente archivo del sistema será el encargado de entrenarlo (*train.m*), siendo éste uno de los dos bloques principales del sistema. El proceso se iniciará siempre y cuando se pulse el

botón destinado a este efecto en el menú principal. Este archivo comprobará si existe una base de datos de logotipos anteriormente creada o si por el contrario debe crear una nueva. Como es una acción bastante compleja, el sistema se asegura de que el usuario cree que se debe realizar el entrenamiento, por lo que se pedirá que se confirme si se desea continuar con el proceso.

Siempre que el sistema no haya sido entrenado con anterioridad se mostrará por pantalla una pequeña ventana para que el usuario confirme que desea seguir adelante con el proceso, como la mostrada a continuación.

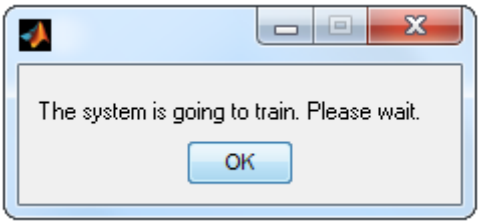


Imagen 17 - Confirmación de entrenamiento 1.

Si por el contrario el sistema ya ha sido entrenado con anterioridad y es necesario sobrescribir los datos ya existentes, se comunicará al usuario que los logotipos anteriormente almacenados serán borrados para dejar paso a los nuevos. Esta última será del tipo siguiente.

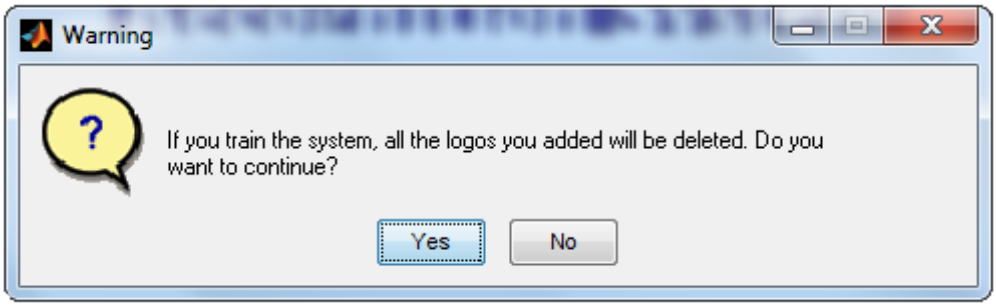


Imagen 18 - Confirmación de entrenamiento 2.

Cuando se pulse el botón *OK* en la primera de las ventanas o *Yes* en la segunda se procederá a la elección de la carpeta que contiene las imágenes para realizar el entrenamiento del sistema. Este proceso, se encargará de realizar un preprocesamiento de las imágenes (comprobar que tiene una extensión válida, está en escala de grises y es del tipo *single*) y de extraer las características principales (puntos clave) de cada imagen y almacenarlas para un posterior reconocimiento. Estas características se extraerán mediante el comando `vl_sift`, anteriormente descrito. Se almacenarán los keypoints y los descriptores en el archivo de características *features*, como un vector de características de imágenes (*storedImages*) en las que cada posición del mismo se corresponde a una cada uno de los archivos visuales de la carpeta elegida.

A medida que se avance en la realización del entrenamiento se mostrará en la ventana de comandos de Matlab el porcentaje de archivos tratados hasta el momento. Cuando se termine de realizar el entrenamiento de todas las imágenes del directorio se mostrará una ventana de información al usuario en la que se mencione que el proceso ha sido completado como el de la siguiente figura:

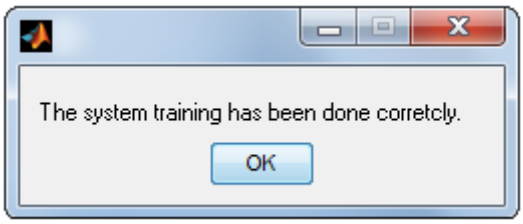


Imagen 19 - Entrenamiento realizado.

Finalmente se activan los botones de reconocer (imágenes y directorios) del menú principal.

El principal archivo necesario para la realización del reconocimiento será el denominado *recognize.m*, que utilizarán tanto el reconocimiento de imágenes como el de directorios. Este archivo se encargará de preprocesar la imagen (comprobar su extensión, conversión a escala de grises y a tipo *single*), de la extracción de las características de la imagen a reconocer y de la comparación (búsqueda de coincidencias) con los puntos clave extraídos durante el entrenamiento. Para realizar dicha comparación, se utilizará la herramienta `vl_ubcmatch`, explicada anteriormente. Tras esta búsqueda se almacenan los resultados obtenidos (*matches* y *scores*) en la variable *results* del archivo *features*. También se realiza una búsqueda de los *scores* menores que un umbral impuesto y se almacenan los que pasen ese filtro en las variables *acceptedScores* y *acceptedMatches*, dentro también de la variable *results.*

Como ya se ha mencionado antes, si el porcentaje de *scores* aceptados supera un umbral impuesto, se considerará que el logotipo buscado está contenido en la imagen y se almacenarán también los valores de los puntos clave en los ejes x e y (*keypointsX y keypointsY*), el nombre del punto clave que aparece (*nameResult*) y las posiciones de los puntos clave situados más arriba y a la izquierda y más abajo y a la derecha (*xMin, xMax, yMin* e *yMax*) para su posterior marcado. Existe también otra variable denominada *existLogo*, que tendrá valor 1 si aparece el logotipo en la imagen y 0 si no aparece, que también se almacenará.

Se puede observar que la variable *results* almacenada en el interior del archivo *features*, contendrá toda la información necesaria para la realización del proceso de marcado en la imagen (o la realización un listado) de los logotipos que aparecen en un cierto archivo visual de entrada.

El siguiente archivo que se deberá implementar es el encargado de realizar el reconocimiento de las imágenes individuales que el usuario introduzca en el sistema. Se denominará a este archivo con el nombre *recognizeImage.m*. Pulsando el botón del reconocimiento de imágenes ("*Recognize the logos of a image*") en el menú principal, se accederá a una nueva interfaz gráfica en la que se podrá elegir una imagen a reconocer. La ventana de reconocimiento individual será del tipo siguiente.

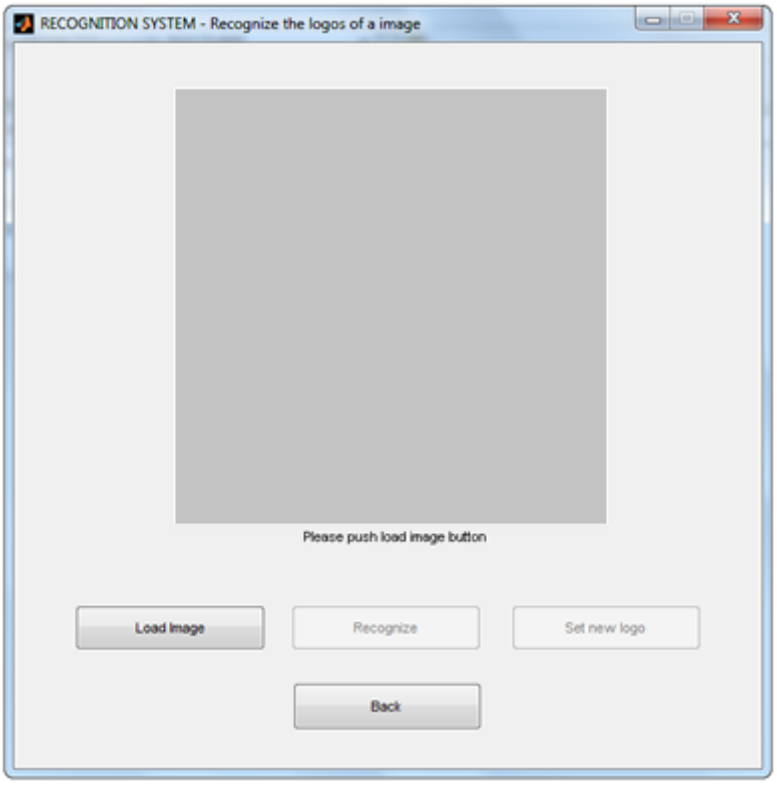


Imagen 20 -Ventana de reconocimiento de imagen individual.

Como se puede observar la ventana contiene cuatro botones, un área destinado para mostrar la imagen y otro para mantener el contacto con el usuario a través de pequeños textos. La única forma de abandonar el proceso de reconocimiento será pulsando el botón "Back", que devolverá al usuario al menú principal. En un primer momento, y como se puede observar, los botones destinados a realizar el reconocimiento ("*Recognize*") y la posibilidad de añadir un nuevo logotipo ("*Set new logo*") estarán desactivados. Es obvio que para realizar el proceso de reconocimiento se deberá cargar una imagen en el sistema, tarea que se llevará a cabo al pulsar el botón de cargar imagen ("*Load image*") y que desencadenará la salida de una ventana para la elección de un archivo por parte del usuario. Una vez cargada la imagen en el sistema, se mostrará en el área reservada y se activa el botón para el reconocimiento ("*Recognize*").

El sistema permanecerá en espera hasta que se pulse el botón de reconocimiento, momento en que se comenzará a buscar los parecidos de la imagen actual con las almacenadas durante el entrenamiento. Para llevar a cabo esta tarea, se utilizará el archivo *recognize.m,* que anteriormente se ha descrito.

Una vez realizada la búsqueda de coincidencias entre la imagen cargada y las almacenadas en el entrenamiento se muestran los resultados en la pantalla, haciendo un recorrido por las variables *exitsLogo* de cada una de las posiciones vector de variables *results* (almacenado en el archivo *features*) que definirán cual (o cuales) logotipos aparecen en dicha imagen.

Siempre y cuando la variable *exitsLogo* tenga valor 1, aparecerá dicho logotipo en la imagen en la que se ha realizado el reconocimiento, momento en el cuál se marcará en pantalla (utilizando los valores *xMin, xMax, yMin, yMax* y *nameResult* para realizar esta tarea), también se activará (aparezca o no aparezca algún logotipo) el botón para añadir nuevos logotipos al sistema ("*Set new logo*")). Un ejemplo del resultado de estas acciones será el siguiente.

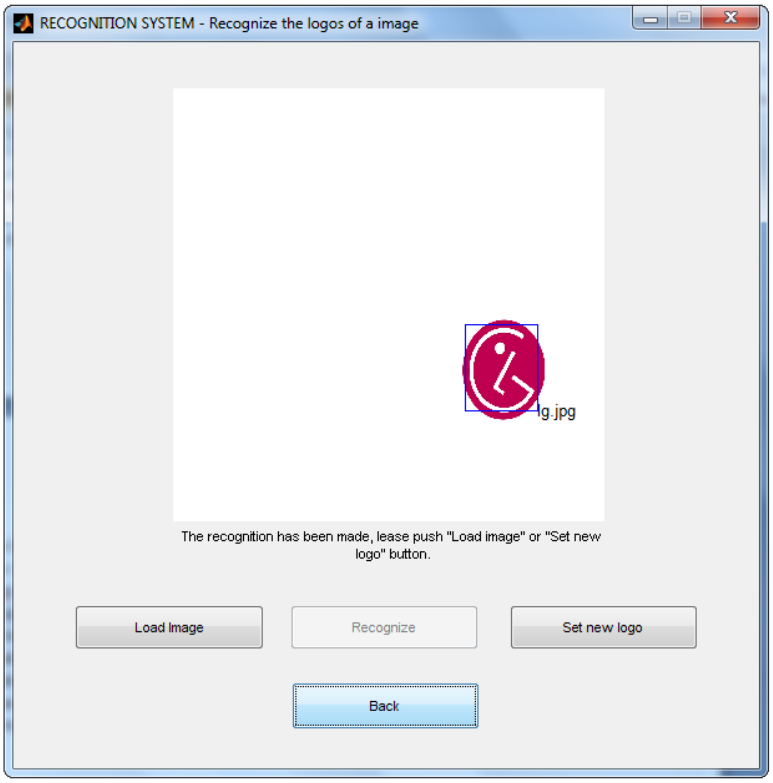


Imagen 21 - Resultados de la búsqueda en imagen individual.

Como puede observarse, el logotipo ha sido marcado mediante un recuadro de color azul y su nombre. También puede verse que el texto del área destinado a la comunicación con el usuario ha cambiado. Si en algún momento el sistema no obtuviera resultado para alguno de los logotipos que aparecieran en la imagen, se da la posibilidad al usuario de añadirlo a los ya existentes. Para realizar esta tarea basta con pulsar el botón "*Set new logo*" y realizar la selección del área donde esté localizado el nuevo logotipo. A continuación, se pide al usuario que seleccione un nombre para el nuevo elemento, y se realiza una comprobación de que el nombre proporcionado no está ente los ya almacenados. Las características (puntos clave y descriptores) del nuevo logotipo se almacenan en la última posición de la variable *storedImages* del archivo *features*.

Otro de los procesos que utilizará el archivo *recognize.m* será el reconocimiento de logotipos de las imágenes contenidas en un directorio. El archivo para la realización de este reconocimiento se denominará *recognizeDirectory.m*. Este tipo de procesado será sumamente útil para realizar un reconocimiento automatizado de una serie de imágenes sin tener que observar los resultados visualmente, pues en este caso el reconocimiento se observará en un fichero de texto que tendrá por nombre RESULTS_TEXT_date_*fecha*_hour_*hora* (donde *fecha* y *hora* serán la fecha y la hora en la que se ha realizado el reconocimiento) y en la ventana de comandos de Matlab. En ambos lugares, se imprimirá el nombre del directorio y a continuación los nombres de las distintas imágenes con un listado de logotipos que aparecen en ellas. Este archivo no tendrá asociada ninguna interfaz gráfica, pues para que el reconocimiento de los archivos sea automático y rápido no es necesaria.

El último de los archivos que contendrá una ventana será el que se encargue de mostrar los créditos del programa. Pulsando el botón "*About*" en la ventana del menú principal se accede a otra interfaz que contiene información acerca del autor del sistema, un botón para observar

unas instrucciones básicas del manejo del mismo ("*View instructions*") y otro para volver al menú inicial ("*Back*"). La ventana asociada a este archivo será la siguiente.

Imagen 22 - Ventana de créditos del sistema.

Tras un diseño exhaustivo de los distintos archivos que serán necesarios para la realización de todas las acciones del sistema de reconocimiento, se puede concluir que serán necesarios siete archivos principales: *startup.m* (encargado de realizar un inicializado de las variables y la creación de la ventana del menú principal), *main.m* (encargado de gestionar la apariencia y las acciones de los botones del menú principal), *train.m* (realización del proceso de entrenamiento del sistema), *recognize.m* (búsqueda de características coincidentes entre una imagen de entrada y las almacenadas durante el entrenamiento), *recognizeImage.m* (reconoce los logotipos que aparecen en una imagen de entrada), *recognizeDirector.m* (reconoce los logotipos que aparecen en las imágenes de un directorio) y *about.m* (encargado de mostrar los créditos y las instrucciones básicas del sistema).

Los dos bloques principales del sistema serán realizados por los archivos *train.m* (proceso de entrenamiento) y *recognize.m, recognizeImage.m* y *recongnizeDirectory.m* (proceso de reconocimiento).

El diagrama de procesos que resume las actividades de los distintos archivos es el siguiente:

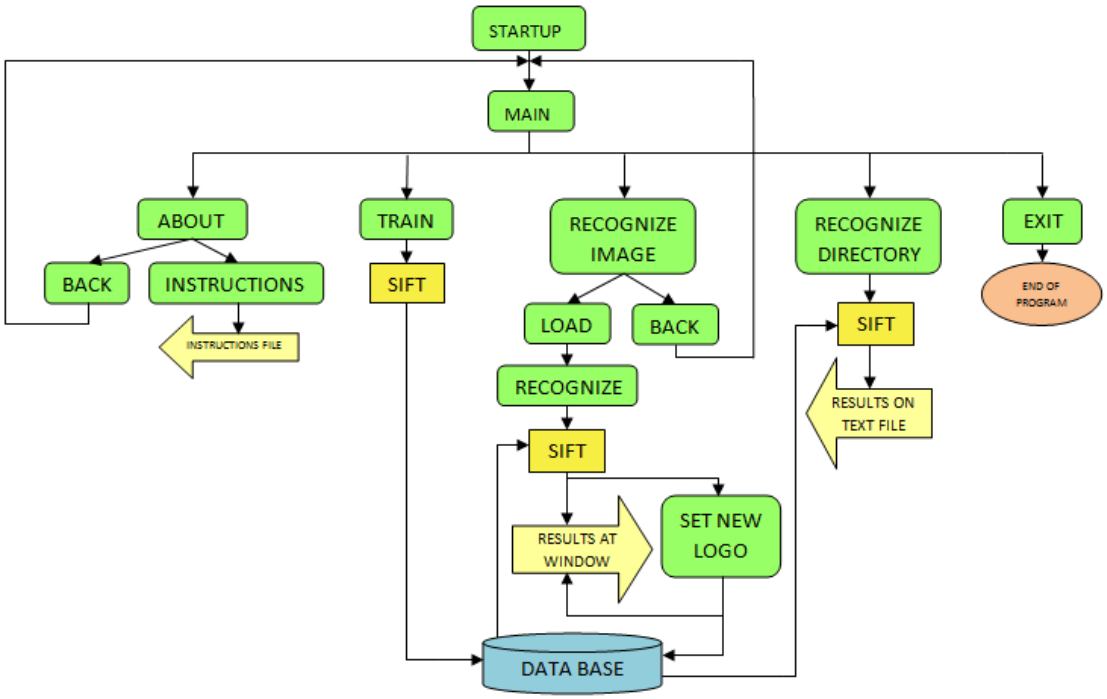


Imagen 23 - Diagrama de procesos del sistema.

Se han obviado las flechas correspondientes a la vuelta al estado de espera del programa en los casos siguientes:

- Durante la aparición del archivo de instrucciones el sistema vuelve a la ventana de créditos (*about*).
- Tras la realización del entrenamiento del sistema (*train*) éste vuelve al menú principal (*main*).
- Tras la realización del reconocimiento de la imagen (*recognize*) o la introducción de un nuevo logotipo al sistema (*set new logo*) el sistema vuelve a la ventana de reconocimiento de imágenes individuales (*recognize image*).
- Una vez realizado el reconocimiento de los archivos del directorio (*recognize directory*), el programa vuelve a la ventana de menú principal (*main*).

Tras la realización del diseño de todos los archivos del programa y las interconexiones entre los mismos, se realiza la implementación que se detallará en el siguiente apartado.

4.2.- IMPLEMENTACIÓN

Tras la etapa anterior, se ha diseñado un sistema que permite reconocer una serie de logotipos (podría reconocer cualquier tipo de imágenes) almacenados durante un entrenamiento.

A continuación se detallan cada uno de los ficheros .m que forman el sistema de reconocimiento, comentando sus aspectos más importantes y cuáles de ellos incluyen una interfaz gráfica.

4.2.1.- startup.m

Mediante la línea *startup* en la ventana de comandos de Matlab, se inicia el programa.
En este archivo de mismo nombre y extensión .m, se realiza como primera opción un borrado de las variables almacenadas ("clear all;") y del texto que pueda haber en la ventana de comandos de Matlab ("clc;").

A continuación, se escribe en la pantalla de comandos que el programa está siendo inicializado con la sentencia "`disp('Texto a mostrar');` " donde se añadirá en el espacio entre las dos comillas simples el texto que se desea que aparezca en la pantalla de comandos de Matlab. En este caso, se comunica al usuario que el programa se está inicializando y que el paquete VL_FEAT se está cargando.

Mediante el código "`run('VL_FEAT-0.9.9/toolbox/vl_setup');`" se carga el paquete de herramientas VL_FEAT que contiene todos los métodos de la transformada SIFT y mediante otro "`disp`" se comunica al usuario que el toolbox ha sido cargado correctamente.

Una vez realizadas estas dos labores, se comprueba si el archivo de características (*features.mat*) está creado. Si no lo está se crea y se guarda. Esta comprobación se realizará mediante el código:

```
if (exist('features.mat')== 2)
    load features.mat;
else
    storedImages = [];
    save features.mat storedImages;
end
```

Tras estas tres tareas, se ejecuta la ventana principal del programa, denominada *main*, que contiene los botones necesarios para realizar todas las actividades que el sistema permite.

4.2.2.- main.m

Tras realizar el arranque del programa, se pasa al menú principal, que constará de la primera de las interfaces gráficas.

En esta ventana, se podrá apreciar un logotipo identificativo del sistema de reconocimiento, un pequeño texto que indicará en cada ocasión una pequeña descripción de las posibles tareas que pueden ser realizadas y una serie de botones, que según sean pulsados realizarán una acción u otra.

Los botones de reconocimiento de logotipos en imágenes o en directorios aparecerán desactivados en el caso de que no se haya realizado un entrenamiento previo del sistema. Para hacer esto, se comprueba si el vector *storedImages* está vacío, y si lo está se desactivan los botones con una sentencia del tipo:

"`set(handles.NOMBRE_DEL_BOTON, 'Enable', 'off');`"

Donde "`NOMBRE_DEL_BOTON`" se sustituirá por los nombres de los botones que se desee desactivar; en este caso los correspondientes a reconocer logotipos en imágenes o en imágenes de un directorio. Para volver a activar posteriormente los botones se utilizará la misma sentencia pero cambiando 'off' por 'on' al final de cada una de ellas.

También se inicializa un texto en el área reservado para los mensajes que el sistema comunica al usuario. Mediante sentencias del tipo siguiente, se podrán cambiar los mensajes de ese área.
"`set(handles.text,'String','Please, choose an option.');`"

Donde "`'Please, choose an option.'`", se sustituirá por los distintos textos que se desea que aparezcan.

A continuación se detallará cada uno de los botones y las acciones que se realizan al ser pulsados:

- El botón **"Train the system"** realiza un entrenamiento del sistema con las imágenes que se desee que posteriormente sea posible reconocer. Si el programa no tiene un entrenamiento previo, los botones de reconocimiento de imágenes y directorio estarán desactivados y no podrán ser pulsados.

  Para realizar esto, se debe leer el archivo de características ("`load features.mat;`").

  Si no existe un entrenamiento previo, el vector *storedImages* estará vacío. En este caso, se esperará a que el usuario lea un mensaje en la pantalla y lo acepte.

  La sentencia "`pause(1);`" se añade para que el sistema realmente espere la confirmación del usuario, pues el sistema no espera si no se activa la pausa y continúa realizando las actividades aún cuando el usuario no ha validado todavía el mensaje que aparece.

```
if (isempty(storedImages))
      uiwait(msgbox('The system is going to train. Please
      wait.'));
      pause(1);
else
      …
end
```

  Si por el contrario, el vector *storedImages* no está vacío, se muestra en pantalla otro mensaje totalmente distinto al anterior, pues las características de las imágenes almacenadas serán borradas para dejar paso a las nuevas.

  El usuario debe pulsar sólo el botón "**Yes**" para continuar realizando el entrenamiento del sistema. En el caso de pulsar "**No**", o de cerrar la ventana, el sistema volverá a la pantalla de menú y no se entrenará.

  El código que gestiona la aparición de este mensaje de advertencia y de qué botón ha sido pulsado es el siguiente:

```
if (isempty(storedImages))
      …
else

      choice = questdlg('If you train the system, all the logos
      you added will be deleted. Do you want to
      continue?','Warning','Yes', 'No', 'Yes');
```

```
        switch choice
        case 'Yes'
        case 'No'
            return;
        case ''
            return;
        end

end
```

Como se puede apreciar, mediante el comando "questdlg" se pregunta al usuario si quiere seguir con el entrenamiento del sistema, borrando los logotipos que habían sido almacenados anteriormente.

Mediante un algoritmo "switch" se observan los distintos casos posibles:
- Si se elige "Yes", no se realiza ninguna acción y el programa continúa.
- Si se elige "No" o una cadena vacía "return" (que será el equivalente a pulsar el botón de cerrar la ventana) el programa utiliza "" para volver al menú principal.

Independientemente de si el sistema había sido o no entrenado con anterioridad, y si se continúa con el desarrollo del programa (pulsando "Ok" en la primera de las pantallas o "Yes" en la segunda), se bloquean las acciones de todos los botones para que el usuario no pueda interrumpir el entrenamiento del sistema y en un algoritmo "try-catch" se realizan todas y cada una de las acciones siguientes.

Se pregunta al usuario la dirección de la carpeta que contiene las imágenes de entrenamiento mediante:

```
pathTrain = uigetdir('', 'Choose the folder of the images to
train the system');
```

Este código dará lugar a la pantalla de la figura 24, para que el usuario escoja la dirección de entrenamiento.

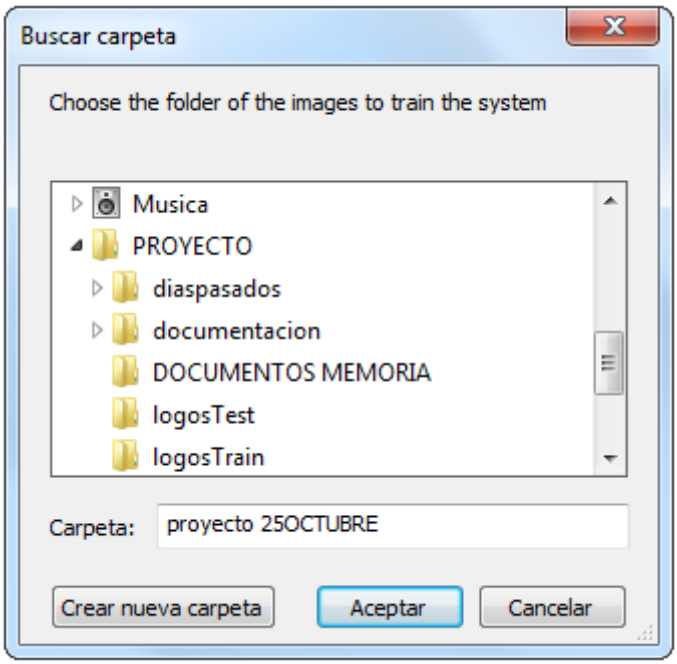


Imagen 24 - Elección de la carpeta de imágenes de entrenamiento.

Siempre que la variable *pathTrain* esté vacía se volverá al menú principal, activando los botones de las funciones que se puedan realizar y cambiando el texto en el área de mensajes. En otro caso se continuará con el programa, comunicando al usuario por la ventana de comandos de Matlab que el sistema está realizando el entrenamiento del sistema.

Llega el momento de realizar el entrenamiento, que en este caso se hará mediante el uso de otro fichero Matlab denominado *train.m* al que se añadirá como parámetro de entrada la variable *pathTrain*, que contiene la dirección de la carpeta de entrenamiento; más tarde se detallará el contenido de este archivo.

Una vez terminado el proceso del archivo *train.m,* se comprueba si el vector *storedImages* continúa estando vacío, o se ha almacenado alguna característica de las imágenes de entrenamiento, de nuevo con la sentencia "`isempty(storedImages)`".

Si dicho vector está vacío se esperará a que el usuario lea un texto en el que se indica que la carpeta elegida no tenía ninguna imagen para entrenar el sistema, y en el que también se le recuerda que pulsando de nuevo el botón *"***Train the system***"* podrá elegir otra carpeta donde existan imágenes para entrenar el sistema.

En el área de mensajes se le insta a que pulse de nuevo el botón de entrenar el sistema. Si el vector contiene alguna característica almacenada, se comunicará en un mensaje emergente y en la ventana de comandos de Matlab que el sistema ya ha sido entrenado correctamente. También se almacenará el vector *storedImages* en el archivo *features.mat* mediante el código:
"`save features.mat storedImages;`"

Como ya ha finalizado el entrenamiento de forma correcta, se activan los botones de reconocimiento.

Tanto si el sistema se ha entrenado, como si no lo ha hecho, se activan los botones de entrenar el sistema, créditos y salida, añadiendo también un mensaje de texto en el área reservado y se espera a que el usuario pulse cualquiera de las acciones posibles.

Como se ha mencionado antes, todas estas acciones están contenidas en un algoritmo "`try-catch`", para que si se obtiene algún error sea posible capturarlo y comunicárselo al usuario de forma correcta.

Si por algún casual se obtuviera un error, se le comunicará al usuario en una ventana de error y por la ventana de comandos de Matlab. Comprobando si se tiene alguna característica almacenada en el vector *storedImages*, se activan los botones en los que se pueda pulsar y se espera a que el usuario vuelva a realizar alguna acción.

- El botón **"Recognize the logos of an image"**, realiza un reconocimiento de los logotipos que aparecen en una cierta imagen. Este botón aparecerá desactivado hasta que no se realice un entrenamiento del sistema.
-

Las únicas acciones que se realizarán en el fichero *main.m* son las de cerrar la pantalla actual (*main*) mediante la sentencia "`closereq;`", y abrir la ventana de reconocimiento de imágenes del programa *recognizeImage*, de la que más tarde se hablará en el apartado del fichero Matlab *recognizeImage.m.*

- El botón **"Recognize the logos of a directory"**, crea un archivo donde se indican los logotipos que aparecen en cada una de las imágenes contenidas en un directorio. Los datos también se muestran en la ventana de comandos de Matlab. Este botón aparecerá desactivado hasta que no se realice un entrenamiento del sistema.

  Como primera acción realizada por este botón, se encuentra la desactivación de todos los botones de la ventana del menú, para evitar que el usuario realice alguna acción indeseada.

  También se añade un texto en el cuadro destinado para ese efecto, donde se comunica al usuario que debe elegir una carpeta donde estén contenidas las imágenes a reconocer.

  En este caso, también se añadirá todo el código en un algoritmo `try-catch`, para capturar los posibles errores que se puedan obtener en el proceso del reconocimiento.

  De nuevo, al igual que al elegir la carpeta de entrenamiento del sistema, se utiliza la sentencia "`uigetdir`" para pedir al usuario la dirección del directorio de imágenes a reconocer. Si el usuario cancela la elección de la carpeta se volverán a activar los botones y se esperará de nuevo su intervención. Si por el contrario, el usuario notifica un directorio correcto, se comunica al usuario la ruta de los ficheros con los que se comprobará el sistema y que los resultados serán visibles en la pantalla de comandos de Matlab y en el fichero creado a tal efecto. En este momento, se pide la ejecución del archivo *recognizeDirectory*, añadiendo como parámetro de entrada la dirección de la carpeta con las imágenes a reconocer. Más adelante se detallarán las acciones realizadas por dicho archivo.

  Tras un correcto reconocimiento, se comunica al usuario que todo el proceso de reconocimiento se ha realizado de forma correcta y dónde puede ver los resultados.
  Se vuelven a activar los botones y se espera una nueva orden del usuario.

- El botón **"About"**, muestra los créditos del programa y permite la lectura de un fichero de texto donde se indican las instrucciones del sistema.

  Se cierra la pantalla actual (*main*) mediante la sentencia "`closereq;`", y se abre la ventana de créditos del programa *about*, de la que más tarde se hablará en el apartado del fichero Matlab *about.m.*

- El botón **"Exit"** permite cerrar el programa. La opción de cerrar la ventana mediante un clic en la cruz de la parte de arriba y a la derecha se ha desactivado para evitar que el usuario cierre la interfaz durante un proceso que no permita que ésta se cierre mediante el código:

  "`function closeGui (src,evnt)`" , dejando vacío el cuerpo de dicha función.

  Por tanto la única forma de acabar el programa será este botón.

  Mediante la sentencia "`closereq;`", se abandona el programa completamente, evitando que cualquier proceso se quede abierto. También se comunica al usuario que

el sistema se está cerrando y se redacta en la pantalla de comandos de Matlab un mensaje agradeciendo al usuario que haya utilizado el programa.

4.2.3.- train.m

Este método, realiza un entrenamiento del sistema, donde se almacenarán las características de las imágenes que luego serán usadas para el reconocimiento.

En este caso, el entrenamiento se realiza con una serie de logotipos, que podrán aparecer en las imágenes que sean introducidas al sistema para su reconocimiento.

Este archivo es llamado desde uno de los botones del menú principal del sistema (*main*), más concretamente desde el botón **"Train the system".**

Si el sistema no tiene un entrenamiento previamente realizado, no será posible realizar el reconocimiento, por lo que en el menú principal se desactivan de forma automática los dos botones que lo permiten realizar.

Tras tener en cuenta estos detalles, en primer lugar se cargan las imágenes de un directorio que el usuario ha elegido, y que en este fichero se añaden como parámetro de entrada. Para cargarlas en el sistema, se utilizará la sentencia:
"`contentPathTrain = dir(pathTrain);`".

Tras cargar el directorio de imágenes, los tipos de imágenes soportados, se almacenan en un vector, que se define de la siguiente forma:
"`supportedImage = {'.BMP'; '.GIF'; '.HDF'; '.JPEG'; '.JPG'; '.PCX'; '.PNG'; '.TIFF'; '.XWD'; '.bmp'; '.gif'; '.hdf'; '.jpeg'; '.jpg'; '.pcx'; '.png';'.tiff'; '.xwd'};`".

Es necesario comprobar la extensión tanto en mayúsculas como en minúsculas, pues pueden darse ambos casos.

Se usará un bucle `for`, para ir leyendo y tratando cada una de las imágenes del directorio.
Se obvia que en un directorio existen dos direcciones ("." y ".."") que no deben ser tenidas en cuenta, para ello se usará el método `isdir`, que devolverá un 0 siempre que el archivo a tratar no sea un directorio.

A medida que se vaya avanzando por el bucle `for`, quedará menos porcentaje de ficheros que tratar, por tanto, se hace necesario conocer la cantidad de ellos que hemos tratado, dando así al usuario una vista general de lo que tardará el sistema en entrenar todas las imágenes. Para ello, se introduce por pantalla un texto que indicará en qué porcentaje del entrenamiento se encuentra el sistema en cada momento. Mediante el siguiente texto se introduce el texto en la ventana de comandos:

```
clc;
disp(' ');
percentage = 100*(i/(length(contentPathTrain(:,1))));
disp(['Training: ' num2str(percentage) ' % completed.']);
```

Tras no tratar los dos directorios, ir mostrando el porcentaje en pantalla y si el archivo que se esté tratando tiene un formato aceptado, se almacena la imagen en la variable *image* leyéndola mediante `imread`.

Antes de realizar la transformada, se debe comprobar que la imagen es del tipo `single` y escala de grises. De no ser así se debe convertir a ese tipo mediante "`rgb2gray`". La forma de convertir las imágenes RGB a escala de grises ha sido comentada anteriormente en el apartado de Diseño.

Tras todas estas comprobaciones, se realiza la transformada SIFT de la imagen que esté almacenada con nombre `image` mediante el uso de la sentencia:
"`[keypoints,descriptors] = vl_sift(image);`".

Esta sentencia forma parte del paquete de herramientas de VL_FEAT y sigue el proceso descrito anteriormente en el apartado 3 (Algoritmo SIFT) y 4 (Diseño).

Si en algún momento se desea realizar una transformada con unos parámetros distintos de pico o borde se usarán las siguientes líneas de código:

```
f = vl_sift(image, 'PeakThresh', VALOR_PEAK) ;
f = vl_sift(image, 'EdgeThresh', VALOR_EDGE) ;
```

Donde *VALOR_PEAK* y *VALOR_PEAK* serán los valores de *pico* o *borde* con los que se realizará la Transformada SIFT

Tras la realización de la transformada, se almacenan los valores de los puntos clave y de los descriptores en sendas variables con nombres *keypoints* `y` *descriptors* respectivamente. Son almacenados también el nombre de la imagen y su tamaño.

Todas estas características se guardan en la variable *storedImages*, desde la que se podrá acceder a ellas posteriormente de una forma muy sencilla poniendo `storedImages(num)`, un punto y el nombre de la variable que se quiera obtener (`sizeX`, `name`, `keypoint`, `etc`), siendo `num` el número de imagen.

Mediante la sentencia `catch`, se captura cualquier error que se haya podido cometer y se le comunica al usuario en la ventana de comandos de Matlab, esperando que éste lea el mensaje en una nueva ventana de error. También damos el valor 1 a una variable creada anteriormente denominada "`haveError`" (con valor 0) que servirá a continuación para ver las acciones que el sistema debe realizar. Si no se obtiene ningún tipo de error la variable "`haveError`" seguirá teniendo valor 0 y se comunicará al usuario que el sistema ha realizado el entrenamiento de forma correcta.

Una vez realizadas todas las transformadas el programa regresa al archivo *main*, que almacenará la variable *storedImages* en el fichero de características *features.mat.*

4.2.4.- recognizeImage.m

Tras realizar un entrenamiento del sistema, se dará la opción de reconocer logotipos en imágenes o en imágenes dentro de un directorio.

El método *recognizeImage*, lleva asociada una de las tres interfaces gráficas del programa. Esta ventana, será sin duda, la más compleja de las tres, puesto que es necesario que el logotipo se marque de alguna forma en la imagen.

Una vez que el botón de reconocer imágenes ha sido pulsado, y dentro del método "`recognizeImage_OpeningFcn`" creado por defecto durante la creación de la interfaz gráfica, se añadirá una serie de sentencias para eliminar la funcionalidad de alguno de los botones (**"Recognize"** y **"Set new logo"**) y eliminar los ejes de la zona donde se mostrarán las imágenes.

Esta segunda sentencia será del tipo "`set(handles.image,'XTick',[],'YTick',[]);`".

La ventana antes de cargar ninguna imagen para su reconocimiento, contendrá un lugar para las imágenes (inicialmente se mostrará en blanco, pues no ha sido todavía cargada ninguna imagen), una caja de texto para escribir mensajes relevantes para el usuario y cuatro botones: **"Load image", "Recognize", "Set new logo"** y **"Back".**

La única forma de abandonar la ventana será pulsando el botón **"Back"**, que devolverá al usuario a la pantalla de menú, ya que la opción de cerrar la ventana con el botón (`X`) ha sido desactivada. En este caso se realiza la desactivación mediante la sentencia:
"`set(handles.figure1,'CloseRequestFcn',@closeGui);`"

A continuación se expondrán las acciones que realiza cada uno de los botones al ser pulsado.

- El botón **"Load Image"** carga la imagen en el sistema y la muestra en la ventana.

  Una vez que ha sido pulsado, se escribe en la ventana de comandos de Matlab que el programa está trabajando y que por favor se espere. Se carga el archivo de características *features.mat.*

  A continuación, se realiza un algoritmo del tipo "`try-catch`", donde se introducirán todas las sentencias necesarias para cargar la imagen y mostrarla por pantalla. Este algoritmo se utiliza por si el programa se encuentra con un error durante su proceso, pueda ser capturado y tratado.

  Con el código "`[fileName,pathName] = uigetfile('*.*','Choose an image to recognize');`", se trata de obtener la dirección de la imagen que queramos tratar. La variable "`fileName`" será el nombre de la imagen en cuestión y "`pathName`" su ruta.

  Mediante este código, se obtiene la pantalla de la figura 25, donde se elegirá la imagen a reconocer.

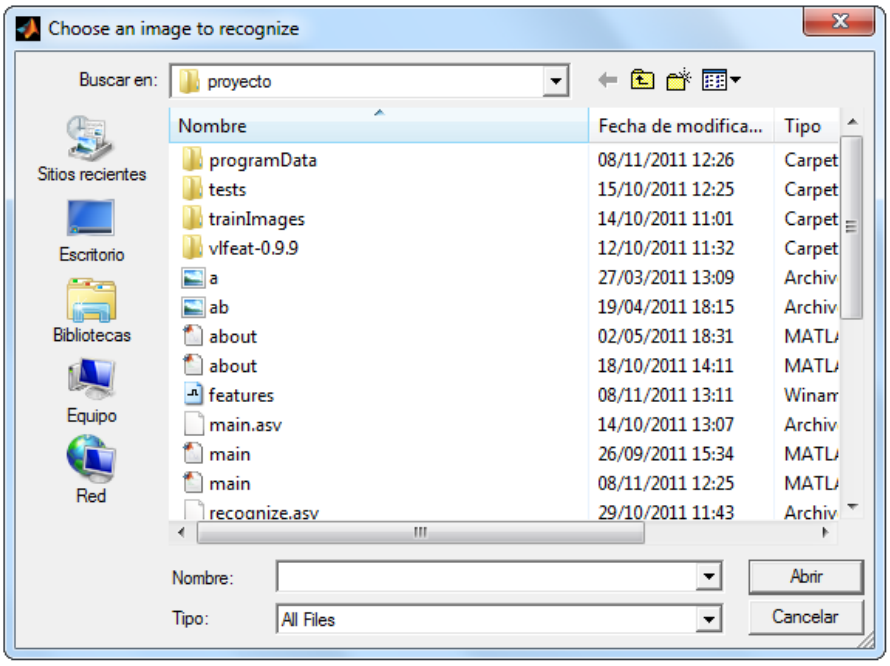


Imagen 25 - Escoja una imagen.

A continuación se desactiva la funcionalidad de todos los botones, para evitar que el usuario los pulse en algún momento que no deba.

Dentro de un bucle `if`, se observa si el nombre del archivo es distinto de 0, puesto que así se habrá elegido una dirección correcta y se podrá cargar la imagen. Las sentencias para realizar esta tarea serán:

```
if isequal(fileName,0)
      …
else
      …
end
```

Siempre que no se elija una ruta correcta (si cumple la primera parte del bucle `if`), se volverán a activar los botones de cargar la imagen y volver al menú. Si por el contrario se elije una dirección válida, se lee y muestra la imagen en pantalla mediante las sentencias:

```
handles.myImage = imread(fullfile(pathName, fileName));
readImage = handles.myImage;
imshow(readImage);
```

Se añade un texto a la ventana de comandos de Matlab y al cuadro de texto donde se indica que el sistema está otra vez listo para realizar cualquier opción.

También se volverá a dar funcionalidad a todos los botones, puesto que cualquiera de las opciones que permiten pueden en este momento ser utilizadas. El único que no podrá ser pulsado es el de almacenar un nuevo logotipo, pues hasta que no se realice el reconocimiento no daremos esa opción.

Por último se guardan las características necesarias en el fichero *features.mat* para su posterior uso, que en este caso serán "*fileName*, *pathName* y *storedImages*".

Si durante este proceso, se obtiene algún error, la parte catch lo capturará y hará las siguientes acciones:

- Se esperará a que el usuario lea un texto de una ventana emergente de error.
- Se escribirá un texto en el cuadro de la pantalla destinado a tal efecto.
- Se volverá a dar funcionalidad a los botones de cargar imagen y volver a la pantalla principal.

En este momento ya se tendrá una imagen cargada y mostrada en la ventana de reconocimiento. El siguiente de los pasos será pulsar el botón de realizar el reconocimiento.

- Tras pulsar el botón **"Recognize"**, se escribe en la ventana de comandos de Matlab y en la caja de texto, comunicando al usuario que el sistema está trabajando. Se desactivan los botones para evitar que el usuario los pulse durante el proceso de reconocimiento. Todas las siguientes tareas se realizarán en el interior de un bloque "`try-catch`".

  Se ejecuta el método *recognize* que soportará toda la carga del reconocimiento. Posteriormente se examinará exhaustivamente dicho archivo.

  Una vez que el reconocimiento ha sido realizado por el método *recognize*, se vuelve a cargar el archivo de características.

  Llega ahora el momento de marcar los resultados en la imagen. Mediante el comando "`hold on`", se abre la opción de pintar sobre la imagen que tenemos en pantalla.

  Si existe la variable de resultados (*`results`*), se realiza un recorrido por todos y cada uno de los resultados obtenidos durante el reconocimiento. En cada uno de ellos se comprueba si la variable "`results(n).existsLogo`" tiene valor 1.

  Si se da el caso, se podrá asegura que ese logotipo está contenido en la imagen que se tiene sobre la pantalla. Llegado ese caso, se procede a crear los vectores necesarios para realizar el marcado del logotipo sobre la imagen y a denominar el texto que se incluirá en la imagen para definir el logotipo existente.

  El siguiente código define los vectores, el texto y lo pinta sobre la imagen:

```
x = [results(t).xMin results(t).xMin results(t).xMax
results(t).xMax results(t).xMin]; % vector de ordenadas
y = [results(t).yMin results(t).yMax results(t).yMax
results(t).yMin results(t).yMin]; % vector abcisas
text (results(t).xMax, results(t).yMax, results(t).nameResult);
plot(x, y);
```

  Tras esto, se escribe en la caja de texto que el reconocimiento ha sido hecho y que se pulse cualquier botón.

  Si en la imagen no se ha reconocido ningún logotipo, se escribe un mensaje de texto indicando que en la imagen actual no se ha reconocido ningún logotipo.

Una vez cerradas las dos posibilidades (se reconoce algún logotipo o no) se cierra la opción de pintar sobre la imagen de la ventana mediante "`hold off`".

Se vuelven a activar todos los botones excepto el de reconocer, y por si alguno de los logotipos que aparecen en la imagen no es reconocido, se da la posibilidad de almacenarlo en el sistema pulsando el botón **"Set new logo"**.

En la parte de captura (`catch`) del bloque "`try-catch`" se capturan los posibles errores que se puedan dar durante el proceso, de esta forma, si se diera algún error se indicaría en una ventana emergente y se activarían todos los botones excepto el de reconocer.

- Botón **"Set new logo".**

Ante la posibilidad de no reconocer alguno de los logotipos que aparecen en una imagen por no estar almacenado en el sistema, se da la posibilidad de almacenarlo para siguientes reconocimientos.

Cuando este botón se pulsa, el fichero de características (*features.mat*) se carga de nuevo para actualizar las variables del sistema.

Posteriormente, se da a elegir si salvar el logotipo en el sistema al usuario mediante el código:

```
choice = questdlg('The logo will be saved in the system. Do you
want to continue?','Warning','Yes','No','Yes');
```

Dando como resultado la ventana de la ilustración 26.

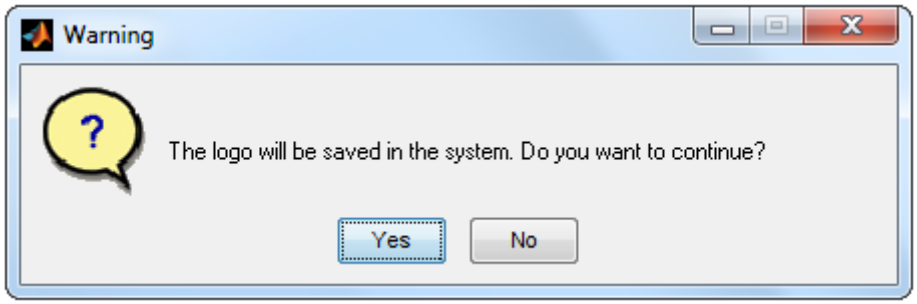


Imagen 26 - ¿Desea guardar el logotipo en el sistema?

Mediante un comando `switch` y observando todos los casos posibles se realizan distintas tareas dependiendo del botón pulsado.

En el caso en que la ventana sea cerrada o se pulse sobre el botón **"No"**, se volverán a activar todos los botones excepto el de reconocer y se incluirá la sentencia "`return`" para que el sistema abandone el algoritmo `switch` y se vuelva al estado anterior al pulsado del botón, esperando una nueva acción del usuario.

En el caso en que se pulse que sí se desea añadir el logotipo al sistema, se desarrollarán una serie de acciones.

La primera de ellas es la desactivación de todos los botones y la inclusión de un texto indicativo de cómo realizar la selección del logotipo mediante:

```
set(handles.text,'String','Please, select the logo and do
double-click over it.');
```

La selección del logotipo se hará mediante: "`selectedArea = imcrop;`".

Donde se tendrá que realizar una selección con el ratón del área del logotipo y hacer doble clic cuando el área deseada haya sido seleccionada. Dicho área se almacenará en una variable denominada "`selectedArea`".

Para que sea mucho más visual, el área seleccionada se muestra en el lugar donde estaba la imagen mediante "`imshow(selectedArea)`".

El siguiente paso será realizar una conversión a `single` y escala de grises (si el área seleccionada no está ya en esos dos formatos).

Se escribe en el cuadro de texto destinado a tal efecto un mensaje indicando al usuario que espere mientras se añade el logotipo al sistema.

Se realiza un bloque "`try-catch`" donde se incluirán diversas acciones que pueden desencadenar algún tipo de error.

A continuación se realiza una lectura del nombre del logotipo que el usuario quiera y se convierte la cadena de caracteres en un `string` mediante "`strcat`".

Todo esto se realiza mediante las sentencias:

```
nameLogo = inputdlg('Write the logo name: ','Write the logo
name');
nameLogo = strcat(nameLogo);
```

La ventana resultante para añadir el nombre del logotipo que se desee almacenar se muestra en la imagen 27.

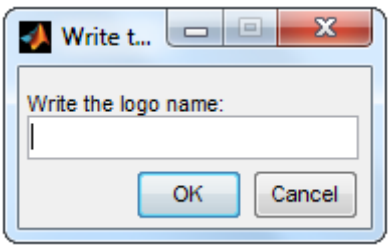


Imagen 27 - Escriba el nombre del logotipo.

Si el nombre introducido está vacío, se cierra la ventana o se pulsa el botón **"Cancel"**, se lanza una ventana emergente en la que se indica que se añada un nombre correcto para almacenar el logotipo. También se activan de nuevo todos los botones, incluido el de reconocimiento. La imagen que se muestra en la pantalla es la que se cargó en un primer momento para su reconocimiento. Tras esto, mediante un "`return`" se vuelve al estado inicial de la imagen antes de su reconocimiento.

Si por el contrario, el nombre introducido es correcto se realiza una comprobación de que el nombre no ha sido utilizado anteriormente en otro logotipo.

Esto se ejecuta en un bucle `for` en el que una por una se comparan los nombres de las imágenes almacenadas con el que se desea dar al nuevo logotipo.

Los nombres de ambos logotipos (almacenados y propuestos por el usuario) se comparan mediante:
`strcmp((nameLogo{1,1}), (storedImages(in).name2))`

Si durante el proceso de comprobación se encuentra un logotipo que tenga el mismo nombre se hace saltar una ventana emergente, como la mostrada en la imagen 26, en la que se indique que ese nombre ya está en uso. Posteriormente se activarían todos los botones y se volvería a leer y mostrar la imagen inicial. Con un "`return`" se vuelve al estado inicial de la imagen antes de su reconocimiento.

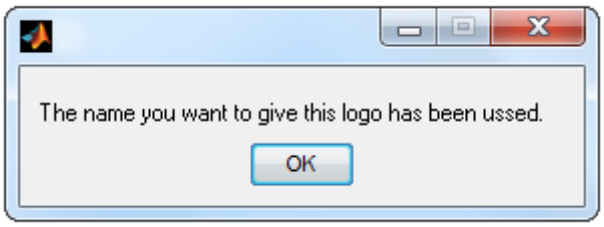


Imagen 28 - El nombre propuesto está siendo utilizado.

También se comprueba que no se pulsa el botón cancel o se sale de la ventana de elección del nombre del nuevo logotipo. En cualquiera de estos casos se obtendría el siguiente mensaje de error:

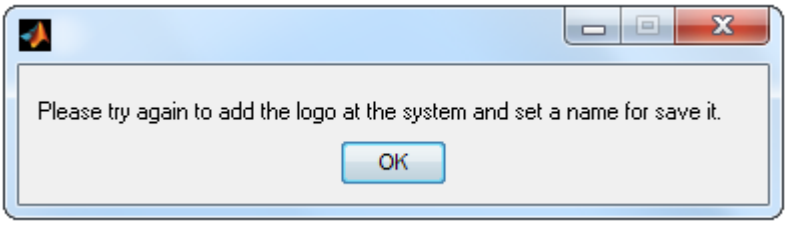


Imagen 29 - Intenta añadir de nuevo el logotipo.

Siempre que no se encuentren nombres coincidentes y se dé un nombre correcto al nuevo logotipo, se almacenan las características al sistema. Para ello, se aumenta el número total de imágenes almacenadas, para que las nuevas características se añadan al final de las que ya se tienen.

Se realiza la transformada SIFT del área seleccionada (que previamente habrá sido convertida a escala de grises y tipo `single`).

Se realiza una normalización de los descriptores dividiéndolos entre 255 para que sus valores estén contenidos entre 0 y 1.

Se almacenan los descriptores, keypoints, tamaño y nombre del nuevo logotipo al vector *storedImages*, y se guarda el archivo *features* para actualizarlo.

Si todos los pasos anteriores se han realizado de forma correcta, se le hace saber al usuario mediante una ventana emergente, igual que la que aparece en la imagen 28 y se activan de nuevo los botones de volver y cargar imagen. Se vuelve al estado inicial antes de realizar el almacenado del nuevo logotipo y a esperar una nueva acción del usuario.

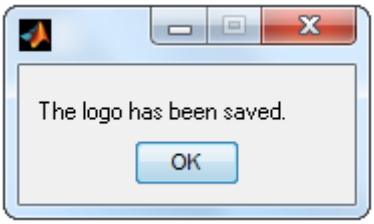


Imagen 30 - El logotipo ha sido guardado correctamente.

Si durante este proceso se ha obtenido algún error, éste será tratado por el `catch`, se lanzará una ventana emergente avisando de ello y se activarán de nuevo los botones de cargar imagen y volver a la pantalla de menú.

Finalmente se escribirá en la ventana de comandos de Matlab que el sistema está listo y se salvarán las nuevas características en el fichero *features.mat*.

- Botón **"Back".**

  En esta pantalla, como ha sido desactivada la opción de cerrar la ventana mediante el botón de arriba y a la derecha, la única forma de terminar el programa sería volver a la pantalla de menú y realizar el cerrado pulsando **"Exit".**

  Cuando el botón **"Back"** se pulsa, se cierra la ventana de reconocimiento y los procesos generados por ella mediante closereq y se abre una nueva ventana del menú principal (*main*).

4.2.5.- recognizeDirectory.m

Tras pulsar el botón de reconocer las imágenes de un directorio en la pantalla de menú saltará una pantalla para elegir el directorio donde se encuentren las imágenes que se desee tratar.

Una vez elegida la carpeta de imágenes, el sistema ejecutará el método *recognizeDirectory*, que realizará un reconocimiento de todos los logotipos contenidos en los archivos del directorio elegido. A este archivo se le añade como parámetro de entrada una variable de tipo string denominada *pathRecognize* que indicará la ruta de la carpeta que contiene los archivos.

Una vez que se esté ejecutando el archivo *recognizeDirectory*, la primera acción que se realizará será la de cargar (mediante la sentencia `load`) el archivo de características *features.mat.*

Se definen los tipos de imágenes soportados y mediante el método `dir`, se observan los archivos contenidos en la carpeta seleccionada. Dicho método devuelve una estructura de tantos elementos como contenga el directorio. Para cada uno de los elementos contenidos en la carpeta, se obtendrán unas variables que indiquen su nombre, la fecha de modificación, el

tamaño (en bytes), si es o no un directorio y una fecha modificada por Matlab que indica su número de serie.

Para la creación del fichero de texto con los resultados y la impresión de los mismos en la ventana de comandos de Matlab se hace necesario obtener los datos de la fecha y la hora en que se realiza el reconocimiento de logotipos.

Mediante el siguiente código, se obtienen unos datos de la fecha y hora en que se realiza la acción y se definen la información de la fecha (*info_date*) y el nombre que tomará el archivo (*info_name*):

```
date = clock;

info_date = ['This document has been created the '
int2str(date(3)) '/' int2str(date(2)) '/' int2str(date(1)) ' at
' int2str(date(4)) ':' int2str(date(5)) ':' int2str(date(6))];

info_name = ['RESULTS_TEXT_date_' int2str(date(3)) '.'
int2str(date(2)) '.' int2str(date(1)) '_hour_' int2str(date(4))
'.' int2str(date(5)) '.' int2str(date(6)) '.txt'];
```

Se crea el documento de texto, donde serán escritos los resultados obtenidos en la realización del reconocimiento. Mediante la función fopen, se crea un archivo con nombre "*RESULTS_TEXT_date_ hour.txt*", donde *date* será la fecha y *hour* la hora a la que haya sido creado el documento.
Posteriormente, y ya habiendo llamado al archivo *text_document*, se irá utilizando la sentencia fprintf para escribir las líneas del fichero de texto. Se utilizarán las letras *wt* para definir que el archivo será del tipo *write*, para poder escribir en él.

Las primeras sentencias utilizadas para abrir y crear el archivo y escribir en la primera línea los datos de la creación del documento y un salto de línea serían las siguientes:

```
text_document = fopen(info_name, 'wt');
fprintf(text_document,'%s',info_date);
fprintf(text_document,'\n');
```

A continuación se realiza el reconocimiento de cada una de las imágenes del directorio. Para evitar errores inesperados, se incluirán las siguientes acciones en un bloque "`try-catch`".

Se recorre el directorio y se comprueba uno a uno si los elementos que se están analizando son o no directorios (mediante la sentencia `isdir`).

Una vez que se comprueba que no es un directorio, se almacena el nombre y la ruta de la imagen que se esté tratando en ese momento y lo se guarda en el fichero de características (*features.mat*), ya que si el tipo de imagen es aceptado se ejecutará el archivo *recognize.m* para que se haga cargo del reconocimiento de la imagen.

Mediante el siguiente código, se observa si el tipo de imagen es alguno de los que el sistema soporta:

```
acceptedImage = 0;
[pathstr, name, extension, version] = fileparts(fileName);
 for j = 1:size(supportedImage,1)
      if strcmp(extension,supportedImage(j,1))
```

```
                acceptedImage = 1;
                break;
        end
end
```

El siguiente paso será comprobar que el valor *acceptedImage* es 1, ya que en ese caso sí se podrá realizar el reconocimiento ejecutando el archivo *recognize* del que más tarde se hablará más detalladamente.

Una vez que el método *recognize* ha realizado su trabajo, se vuelve a cargar el fichero de características con la sentencia "`load features.mat`".

Se inicializan varias variables que más tarde harán falta y se va recorriendo el fichero de resultados para la imagen que se esté procesando.

Siempre que la variable *existsLogo* sea 1, y como se recorrerá la variable de resultados para todas las posibles apariciones de imágenes, se comprueba si es el primer logotipo que aparece en la imagen o no, pues no se desea escribir la primera línea varias veces, puesto que este texto sólo indica una línea general que pondrá en posición sobre qué imagen es la que se está tratando, la posición que ocupa en el directorio, etc. Una vez que aparezca alguno de los logotipos, se cambia a 1 el valor de la variable *firstLineText* (que se habrá inicializado anteriormente a 0).

El algoritmo que comprobará si ha sido escrita la primera línea (tanto en el archivo de texto como en la ventana de comandos de Matlab) será el siguiente:

```
if (firstLineText == 0)
      disp('')
      fprintf(text_document,'\n');

      text = ['The image # ' int2str(images) ' of the directory,
      with name: ' name ' contens the following logos: '];

      disp(text);
      fprintf(text_document,'%s',text);
      fprintf(text_document,'\n');
      firstLineText = 1;
end
```

Cada vez que se esté en una nueva posición del vector de resultados se comprobará si se ha escrito o no la primera línea y a continuación se añadirá, siempre que ese logotipo aparezca en la imagen, el nombre de dicho logotipo. De esta forma, se tendrá para cada imagen que el sistema ha tratado de reconocer una línea de texto donde se indique la posición y el nombre del fichero; y justamente debajo los logotipos que aparecen en dicha imagen (uno en cada línea).

Para añadir el nombre del logotipo que aparezca en un momento dado se buscará la variable *results(n).nameResult*, donde n será la posición del vector de resultados en que la que se esté.
Si por el contrario no existiera ningún logotipo en la imagen que se esté tratando, en lugar de poner una lista con los logotipos que están contenidos en dicha figura, se añadirá un texto que indique que en dicho archivo no existe ningún logotipo de los que el sistema tiene almacenado. En este caso, y como el método *recognizeDirectory* ha sido creado para tratar directorios enteros sin la necesidad de ayuda del usuario, no es posible añadir cualquier logotipo de los que aparezcan y el sistema no tenga almacenado.

Para hacer esto último, se debería cargar la imagen en la ventana de reconocimiento de imágenes individuales, reconocerla y tras no obtener el resultado deseado pulsar el botón de añadir un nuevo logotipo.

Si el directorio no contiene ninguna archivo, o ninguna imagen con un formato válido, la variable *images* tendrá valor 0 y se escribirá en el fichero de resultados y en la ventana de comandos de Matlab que dicho directorio no tiene ninguna imagen válida para reconocer.

En este momento, se cierra el bloque "try-catch" donde se capturan los posibles errores que se hayan podido obtener. En caso de que se obtenga cualquier error durante el proceso de reconocimiento, se escribe en la ventana de comandos que el método *recognizeDirectory* ha tenido un error, se guarda el archivo de características y se cierra el fichero de texto que se está escribiendo para los resultados mediante: `fclose(text_document)`. En caso de que no se obtenga ningún error durante el proceso de reconocimiento se guarda el fichero de características, se cierra el fichero de texto que se está escribiendo y se añade en la ventana de comandos que el reconocimiento ha sido finalizado con éxito.

Tras realizar todas estas acciones, se devuelve el control del programa al menú principal que esperará que cualquiera de los botones sea pulsado por el usuario.

4.2.6.- recognize.m

Este archivo no tiene asociada ninguna interfaz gráfica, pero es el más importante de todo el programa, pues mediante su uso el sistema es capaz de realizar un reconocimiento de la imagen que se esté tratando. Este método se usará en los archivos *recognizeImage.m* y *recognizeDirectory*.m.

Lo primero que se realiza será una carga del archivo de características (*features.mat*). Después se define la imagen leída en una variable (*readImage*), obtenida tras realizar un `imread` del archivo que se desee tratar en ese momento. Se comprueba también que la imagen está en escala de grises y es del tipo `single`. Se realiza la Transformada SIFT y se normalizan los valores de los puntos clave, como ya se comentó anteriormente en el apartado de Diseño.

Se pide el número de imágenes almacenadas durante el entrenamiento y se inicializa un vector para los resultados, denominándolo *results*.

A continuación se buscan coincidencias entre los keypoints obtenidos para la imagen tratada y los almacenados anteriormente en el entrenamiento del sistema mediante la sentencia siguiente:

```
[results(k).matches, results(k).scores] = vl_ubcmatch(descriptors,
storedImages(k).descriptor, 1.5);
```

Donde k es el número de imagen almacenada en la que el sistema se encuentre en cada momento y el valor 1,5 es el umbral que se impondrá para multiplicar la distancia obtenida entre dos descriptores y luego compararla con la distancia del primer descriptor a todos los demás; cambiando ese valor obtendremos unas coincidencias que tendrán una distancia euclídea mayor o menor con el resto de coincidencias para ser aceptadas, dependiendo de si el umbral es más alto o más bajo.

Una vez realizada la búsqueda de coincidencias, llega el turno de realizar un filtrado de los resultados obtenidos.

Como ya se comentó en el Diseño, se impone un umbral para los valores del vector `scores`. Este umbral se decidirá más adelante en el momento de la realización de las pruebas del programa y en función de los resultados obtenidos con los cambios de dicho valor.

Se va recorriendo el vector de `scores` y comprobando uno a uno si es menor que el umbral. En ese caso se guardarán en otras nuevas variables (dentro de *results*) los *scores*, *matches* y *keypoints* que pasen por el filtrado. Tras esta comprobación, ya se tendrá un nuevo listado filtrado, que servirá a continuación para decidir si existe o no un logotipo en la imagen.
La variable definida anteriormente como *acceptedResults*, irá aumentando su valor a medida que se acepten los scores que cumplan la condición del umbral impuesta.

Tras realizar el filtrado, se vuelve a comprobar si existe el logotipo almacenado con el que se esté comparando la imagen de entrada. Para realizar esto, se observará si la cantidad de resultados aceptados tras la búsqueda de coincidencias (*acceptedResults*), es mayor que un porcentaje de los puntos clave que tengan almacenados en la imagen de entrenamiento.

El valor del porcentaje que se usará será definido también durante las pruebas del programa y se expondrá en el apartado de conclusiones.

Siempre que cumpla el umbral del filtrado y del porcentaje de puntos clave coincidentes, se podrá afirmar que existe ese logotipo en la imagen que se esté tratando.

Para ello se impondrá el valor 1 en una variable, contenida dentro del vector *results*, y denominada *existsLogo*. De esta forma, mediante la comprobación de que el valor de *existsLogo* es 1, se podrá conocer rápidamente si existe ese logotipo en la imagen que se haya tratado.

Se almacenan los valores máximo y mínimo de las posiciones de los puntos clave para ambas direcciones del espacio (x e y), que serán utilizados más adelante para marcar el logotipo en la pantalla (en el caso de que se realice el reconocimiento en imágenes, no en directorios).

Se archivarán también el nombre de la imagen de entrenamiento que aparezca en la imagen actual, para conocer el nombre del logotipo que aparece.

Por último se salva el archivo de características, guardando así los resultados obtenidos durante la realización del reconocimiento para su posterior uso o comprobación.

En el caso de que las coincidencias entre la imagen tratada y almacenada no superen el umbral impuesto, se almacenarán las características (nombre, posiciones máximas y mínimas, si existe o no logotipo, etc) con valores 0 o nulos. Y se guardará el archivo de características para su posterior uso o comprobación.

Todas estas acciones se realizan sin tener en cuenta que se pueden obtener errores en su realización, puesto que ya se tendrá en cuenta desde el lugar donde llamemos a este método.

4.2.7.- about.m

Si se pulsa el botón ABOUT en la ventana principal, ésta se cierra y se abre otra que muestra los créditos del programa y un archivo de instrucciones que detalla cómo utilizar el sistema. Aparece una pantalla en la que se muestra el nombre del programa y los créditos del mismo, en este caso nombre, titulación y universidad.

Existen también dos botones en la ventana, el de vuelta al menú principal (**"Back"**) y el de mostrar las instrucciones del programa.
Si se pulsa el botón **"Back"**, se cierra la ventana de créditos mediante `closereq` y se abre de nuevo la ventana del menú principal *main*.
Si por el contrario se pulsa el botón de las instrucciones, contenido en un bloque " `try-catch`", se trata de abrir un fichero de texto mediante la sentencia: `winopen('programData/instructions.txt');`. Si por algún motivo no se pudiera abrir dicho archivo, el error se capturaría y se esperaría a que el usuario leyera un mensaje de error de apertura del fichero de instrucciones.

En este caso, también se realiza un eliminado de la función del cerrado de la ventana con el botón de arriba y a la derecha mediante la sentencia:
```
set(handles.figure1,'CloseRequestFcn',@closeGui);
```

5.- PRUEBAS Y CONCLUSIONES

5.1.- Pruebas realizadas

Como ya se comentó en el apartado de diseño son muchas las maneras de comprobar el funcionamiento del sistema. En este caso se ha contado con una galería de 300 imágenes de 100 logotipos diferentes, es decir, tres imágenes por logotipo. Para realizar el entrenamiento del sistema se han utilizado dos imágenes para cada logotipo y para la comprobación del funcionamiento se han utilizado las 100 restantes. A continuación se observa una muestra de alguna de las imágenes que compondrán el grupo de entrenamiento.

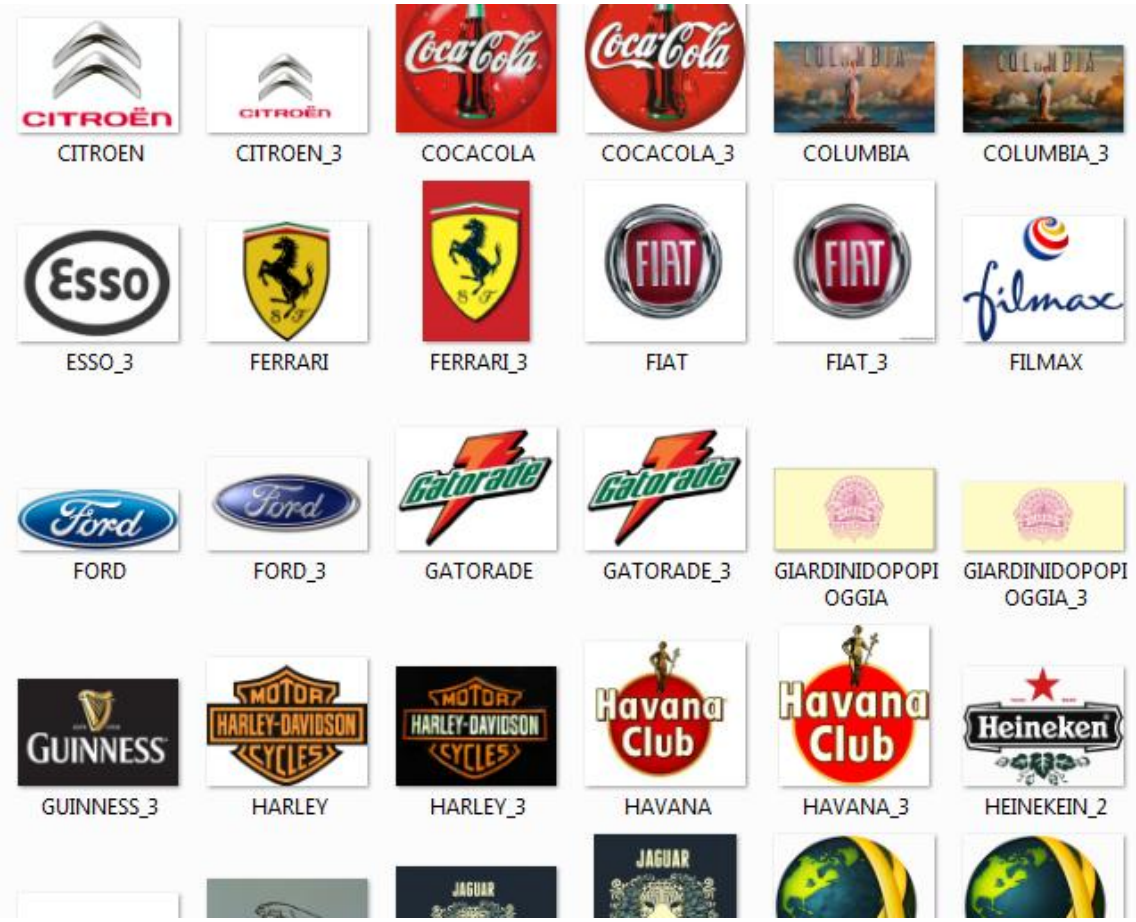


Imagen 31 - Imágenes de entrenamiento.

Para realizar de distintas formas el test de reconocimiento se han dividido estas 100 imágenes en ocho grupos diferentes:

- 13 imágenes individuales. Se han utilizado los logotipos aislados, para comprobar el funcionamiento del sistema ante imágenes individualizadas y sin posibles confusiones.

- 11 imágenes con cambios de escala. Imágenes con ampliaciones o reducciones para comprobar el funcionamiento ante este tipo de cambios.

- 13 imágenes con giros de menos de 30 grados. Como se hablaba en el apartado del Algoritmo SIFT, este tipo de prueba fue una de las que llevaron a la conclusión de que el sistema trabajaba mejor con giros menores de 30 grados, y por lo tanto es un buen método para probar la eficiencia de nuestro reconocedor.

- 12 imágenes con giros mayores de 30 grados. Se debe comprobar el grado de aciertos y fallos con ese tipo de rotaciones.

- 13 imágenes con distinta iluminación. Jugando con el brillo y el contraste de las imágenes de entrada se realizará este testeado del sistema para observar su funcionamiento.

- 13 imágenes ocluidas. Se utilizarán distintos grados de oclusión para comprobar el sistema ante este tipo de imágenes.

- 7 imágenes múltiples. Se comprobará si el sistema es capaz de detectar todos los logotipos (2 por imagen).

- 11 imágenes en contexto. Se han realizado una serie de inclusiones de los logotipos en distintos escenarios, para observar el comportamiento del sistema en escenarios reales con imágenes complejas.

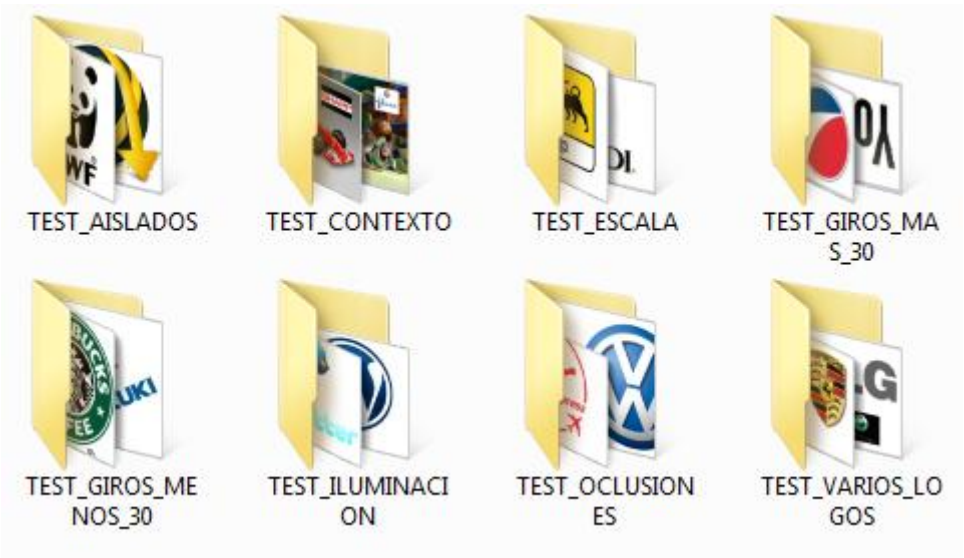


Imagen 32 - Carpetas de imágenes de test.

Para el reconocimiento de los distintos grupos de imágenes, se realizará una comprobación de tres valores, que podrán definir cuán bueno (o malo) es el sistema implementado. Para ello se usarán tres variables: *True Positives (TP),* que son los aciertos del sistema en el proceso de reconocimiento; *False Positives (FP),* que definirán las equivocaciones que tiene el reconocedor apuntando a logotipos que realmente no están presentes en la imagen; *False Negatives (FN),* que serán aquellas omisiones de marcado de los logotipos que están en la imagen. Se calcularán primero estas variables para cada uno de los logotipos, obteniendo los

valores *TP, FN* y *FP* asociados a cada uno de ellos. Después se realizará un sumatorio de todas las variables para obtener los valores globales para cada una de las categorías.

Los resultados se expresarán en las llamadas "*Tablas de contingencia*". En ellas habrá cuatro cuadrantes en los que se colocarán los resultados obtenidos (*True Positives, False Positives y False Negatives*), los denominados *True Negatives* no se utilizarán, pues es absurdo considerar que el sistema acierte cuando no marque ningún logotipo que no aparezca. Como cabe esperar, se desearán obtener unos niveles de True Positives muy altos y de False Positives y False Negatives muy bajos. El aspecto de las tablas es el que se muestra a continuación.

Tabla 1 - Modelo de Tabla de Contingencia.

TABLA DE CONTINGENCIA

		Detección del algoritmo T	Detección del algoritmo F
Realidad	T	TP	FN
	F	FP	TN

A partir de la tabla de contingencia se podrán obtener tres valores que detallarán de forma fiable los resultados obtenidos. Estos valores se denominarán:

- *Precisión* (*P*). De todos los logotipos que el sistema devuelve qué porcentaje de ellos aparecen verdaderamente en las imágenes de entrada. La fórmula para calcular este valor será la siguiente:

$$P = \frac{TP}{TP+FP} \tag{23}$$

- *Cobertura (Recall* (*R*)). Qué porcentaje de logos se obtienen respecto a los que debería obtener. La fórmula asociada al cálculo de este valor es:

$$R = \frac{TP}{TP+FN} \tag{24}$$

- *Medida-F1 (F1)*. Es la ponderación entre los dos valores anteriores. Se calcula de la siguiente manera:

$$F1 = \frac{2 \cdot P \cdot R}{P+R} \tag{25}$$

Se utilizará el método *recognizeDirectory* para realizar un reconocimiento automatizado y se realizarán cambios en distintos umbrales y métodos para la comprobación del funcionamiento

del sistema. Los dos umbrales que se utilizarán en este caso serán: el que define el filtro de resultados aceptados tras el proceso de la búsqueda de coincidencias (*results.scores* deberá ser menor que ese umbral) y el que determina qué porcentaje de puntos clave aceptados para cada imagen debe tener para considerar que un logotipo aparece en dicha escena.

El sistema tarda en comparar la imagen con cada una de las que están almacenadas en el entrenamiento una media de 0,3 segundos, por lo que dependiendo del número de imágenes contenidas en cada tipo de escenas a reconocer se tardará más o menos.

Se realizarán cinco pruebas distintas, en las que se calcularán los valores medios de las tres variables (*P, R y F1*) obtenidos para comprobar los resultados de una forma directa. Estas pruebas se nombrarán con letras de la A a la E para pintar los resultados obtenidos en una gráfica. La equivalencia de las letras y las pruebas se expondrá en el apartado de Conclusiones.

5.1.1.- Prueba con filtrado de scores y de puntos clave aceptados

Durante la primera prueba que se realizará para determinar los umbrales, se usará el método vl_sift para la realización de la Transformada SIFT y el método vl_ubcmatch para la búsqueda de puntos clave parecidos entre las dos imágenes (la de entrada y las almacenadas durante el entrenamiento).

A continuación, se hará un filtrado de los scores obtenidos durante la búsqueda de coincidencias donde solamente se conservarán los resultados menores a 1,5. Si tras este filtrado, todavía existen al menos el 50% de los puntos clave de la imagen almacenada durante el entrenamiento se considerará que ese logotipo aparece en la imagen.

Los resultados obtenidos para esta realización fueron los siguientes:

- Imágenes individuales.
  Utilizando las 13 imágenes (que se compararon con todas y cada una de las que ya estaban almacenadas durante el entrenamiento) se obtuvieron los valores contenidos en la Tabla 2.

Tabla 2 – Tabla de Contingencia imágenes individuales (scores<1,5 y más del 50% de puntos clave).

TP = 7	FN = 6
FP = 4	---

Los valores de *Precisión, Cobertura* y *F1* serán los siguientes:
P = 7/(7+4) = 0,6363 (63,63%)
R = 7/(7+6) = 0,5385 (53,85%)
F1 = (2·0,636·0,5385)/(0,6363+0,5385) = 0,5833 (58,33%)

- Imágenes con cambios de escala.
  Se utilizan 11 imágenes para realizar esta prueba. Los resultados obtenidos se muestran en la tabla siguiente.

Tabla 3 - Tabla de Contingencia imágenes con cambios de escala (scores<1,5 y más del 50% de puntos clave).

TP = 5	FN = 6
FP = 5	---

Los valores de *Precisión, Cobertura* y *F1* serán los siguientes:
P = 5/(5+5) = 0,500 (50,00%)
R = 5/(5+6) = 0,4545 (45,45%)
F1 = (2·0,5·0,4545)/(0,5+0,4545) = 0,4762 (47,62%)

- Imágenes con giros de menos de 30 grados.
  Se utilizan 13 imágenes para realizar esta prueba. Los resultados obtenidos se muestran en la tabla siguiente:

Tabla 4 - Tabla de Contingencia imágenes con giros menores a 30 grados (scores<1,5 y más del 50% de puntos clave).

TP = 6	FN = 7
FP = 14	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 6/(6+14) = 0,300 (30,00%)
R = 6/(6+7) = 0,4615 (46,15%)
F1 = (2·0,3·0,4615)/(0,3+0,4615) = 0,3636 (36,36%)

- Imágenes con giros de más de 30 grados.
  Se usaron 12 imágenes para realizar esta prueba. Los resultados obtenidos se muestran en la tabla siguiente:

Tabla 5 - Tabla de Contingencia imágenes con giros mayores a 30 grados (scores<1,5 y más del 50% de puntos clave).

TP = 5	FN =7
FP = 27	---

Los valores de *Precisión, Cobertura* y *F1* para estos valores fueron los siguientes:
P = 5/(5+27) = 0,1562 (15,62%)
R = 5/(5+7) = 0,4166 (41,66%)
F1 = (2·0,1562 ·0,4166)/(0,1562 +0,4166) = 0,2272 (22,72%)

- Imágenes con distinta iluminación. Usando las 13 imágenes para esta prueba se obtuvieron los siguientes resultados.

Tabla 6 - Tabla de Contingencia imágenes con distinta iluminación (scores<1,5 y más del 50% de puntos clave).

TP = 7	FN = 6
FP = 24	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 7/(7+24) = 0,2258 (22,58%)
R = 7/(7+8) = 0,5384 (53,84%)
F1 = (2·0,2258·0,5384)/(0,2258+0,5384) = 0,3181 (31,81%)

- Imágenes con oclusiones. Mediante la comprobación del reconocimiento de las 13 imágenes con oclusiones parciales se obtuvo lo siguiente.

Tabla 7 - Tabla de Contingencia imágenes con oclusiones (scores<1,5 y más del 50% de puntos clave).

TP = 5	FN = 8
FP = 20	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 5/(5+20) = 0,200 (20,00%)
R = 5/(5+8) = 0,3846 (38,46%)
F1 = (2·0,2·0,3846)/(0,2+0,3846) = 0,2631 (26,31%)

- Imágenes múltiples. Se comprobará si el sistema es capaz de apuntar hacia todos los logotipos que aparecen en las siete imágenes con 14 logos. Se obtienen los siguientes resultados:

Tabla 8 - Tabla de Contingencia imágenes múltiples (scores<1,5 y más del 50% de puntos clave).

TP = 7	FN = 7
FP = 39	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 7/(7+39) = 0,1522 (15,22%)
R = 7/(7+7) = 0,500 (50,00%)
F1 = (2·0,1522·0,5)/(0,1522+0,5) = 0,2333 (23,33%)

- Imágenes en contexto.
  En este caso se obtienen mediante las 11 imágenes que forma el conjunto los siguientes resultados.

Tabla 9 - Tabla de Contingencia imágenes en contexto (scores<1,5 y más del 50% de puntos clave).

TP = 8	FN = 3
FP = 355	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 8/(8+355) = 0,0220 (2,20%)
R = 8/(8+3) = 0,7272 (72,72%)
F1 = (2·0,022·0,7272)/(0,022+0,7272) = 0,0427 (4,27%)

- Valores medios.

A continuación se muestra una tabla con todos los valores P, R y F1 de cada una de las categorías.

Tabla 10 - Resumen de valores P, R y F1 (scores<1,5 y más del 50% de puntos clave).

TIPO DE IMAGEN	P	R	F1
Individuales	0,6363	0,5385	0,5833
Con cambios de escala	0,5000	0,4545	0,4762
Con giros < 30 grados	0,3000	0,4615	0,3636
Con giros > 30 grados	0,1562	0,4166	0,2272
Con cambios de iluminación	0,2258	0,5384	0,3181
Con oclusiones parciales	0,2000	0,3646	0,2631
Múltiples	0,1522	0,5000	0,2333
En contexto	0,0220	0,7272	0,0427
MEDIA	**0,2741**	**0,5002**	**0,3134**

5.1.2.- Prueba con umbral de vl_ubcmatch y porcentaje de puntos clave aceptados

Tras la desastrosa primera prueba, se determinó que no era necesario realizar un filtrado de los resultados obtenidos, pues el propio algoritmo de búsqueda (vl_ubcmatch) ya tenía esta posibilidad. Introduciendo un número mayor que cero al final de la expresión que realiza la búsqueda de coincidencias, es posible obtener sólo los resultados que tienen distancias euclídeas mayores que un cierto umbral entre el descriptor que se esté tratado y todos los demás. Este algoritmo tiene prefijado un valor de 1,5 para dicho umbral. De esta forma, se obtiene un filtrado de los resultados obtenidos, escogiendo aquellos que cumplen la distancia impuesta.
Así pues, el único valor que se puede variar para determinar los umbrales óptimos de la búsqueda de coincidencias será el porcentaje de características coincidentes entre la imagen de entrada y las almacenadas para considerar la aparición de los distintos logotipos.

Teniendo en cuenta también el prolongado tiempo de procesado de las imágenes de test para la realización de la primera de las pruebas, se determina escoger aleatoriamente dos imágenes

para cada una de las categorías (individuales, giros de menos de 30 grados, en contexto, etc) con un total de 18 logotipos, y un conjunto de 42 elementos (2 imágenes por logotipo) para la realización del entrenamiento del sistema.

Utilizando los nuevos conjuntos de entrenamiento y test, e imponiendo un nivel de al menos el 85% de características para considerar que en la imagen aparece alguno de los logotipos almacenados se obtuvieron estos resultados.

Tabla 11 - Tabla de Contingencia global (valor 1,25 en vl_ubcmatch y más del 85% de puntos clave).

TP = 8	FN = 10
FP = 57	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 8/(8+57) = 0,123 (12,23%)
R = 8/(8+10) = 0,4444 (44,44%)
F1 = (2·0,123·0,4444)/(0,123+0,4444) = 0,1926 (19,26%)

5.1.3.- Prueba con umbrales de vl_ubcmatch, pico y borde iguales y porcentaje de puntos clave aceptados

Teniendo en cuenta que existen dos elementos pertenecientes a la categoría de imágenes en contexto, los resultados obtenidos en el apartado anterior en los valores FP no son del todo malos, aunque se piensa en otro método más para refinar la búsqueda de coincidencias entre imágenes, pues aún se obtienen unos niveles FN demasiado altos. Se observó que era necesario obtener un número de puntos clave menores y más precisos, por lo que se concluyó que los factores de *pico* y *borde*, descritos en el apartado de Diseño podrían jugar un papel importante a la hora de afinar la búsqueda de coincidencias y por lo tanto de obtener unos mejores resultados de *Precisión*, *Cobertura* y *F1*.

Se usaron para cada valor del *porcentaje* de puntos clave aceptados para que se considere que existe un logotipo, distintos valores de los niveles de *pico* y *borde* en los procesos de extracción de características del entrenamiento y testeado del programa.

Los resultados obtenidos para los niveles de *pico* = 10 en el entrenamiento y el reconocimiento, un *porcentaje* de más del 50% de puntos clave aceptados para considerar que existe un logotipo y el umbral de vl_ubcmatch a 1,25 fueron los siguientes.

Tabla 12 - Tabla de Contingencia global (nivel de pico 10, vl_ubcmatch 1,25 y más del 50% de puntos clave).

TP = 13	FN = 5
FP = 99	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 13/(13+99) = 0,1160 (11,6%)
R = 13/(13+5) = 0,7222 (72,22%)
F1 = (2·0,116·0,7222)/(0,116++0,7222) = 0,1998 (19,98%)

Manteniendo los valores para los umbrales de vl_ubcmatch y el *porcentaje* de puntos clave aceptados se realizaron también las pruebas para un nivel de *borde* de valor 5, tanto en el entrenamiento como en el test, obteniendo los siguientes resultados.

Tabla 13 - Tabla de Contingencia global (nivel de borde 5, vl_ubcmatch 1,25 y más del 50% de puntos clave).

TP = 13	FN = 5
FP = 92	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 13/(13+92) = 0,1238 (12,38%)
R = 13/(13+5) = 0,7222 (72,22%)
F1 = (2·0,1238·0,7222)/(0,1238+0,7222) = 0,2111 (21,11%)

A continuación se realizaron las pruebas manteniendo el nivel de vl_ubcmatch y el porcentaje de características aceptadas para considerar logotipo, pero realizando una extracción de puntos clave mucho más severa, utilizando un nivel de *pico* de valor 25. La tabla de Contingencia y valores *P, R y F1* obtenidos fueron los siguientes.

Tabla 14 - Tabla de Contingencia global (nivel de pico 25, vl_ubcmatch 1,25 y más del 50% de puntos clave).

TP = 9	FN = 9
FP = 72	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P =9 /(9+72) = 0,1111 (11,11%)
R =9 /(9+9) = 0,5 (50,00%)
F1 = (2·0,1111 ·0,5)/(0,1111 +0,5) = 0,1818 (18,18%)

Utilizando ahora un nivel de borde de valor 2.5, y manteniendo los niveles de los otros dos umbrales se obtuvieron los siguientes valores.

Tabla 15 - Tabla de Contingencia global (nivel de borde 2,5, vl_ubcmatch 1,25 y más del 50% de puntos clave).

TP = 13	FN = 5
FP = 95	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 13/(13+95) = 0,1203 (12,03%)
R = 13/(13+5) = 0,7222 (72,22%)
F1 = (2·0,1203 ·0,7222)/(0,1203 +0,7222) = 0,2062 (20,62%)

Si se realiza una media de estas cuatro pruebas se obtienen los siguientes valores:
P = 0,1178 (11,78%)
R = 0,6666 (66,66%)
F1 = 0,1997 (19,97%)

5.1.4.- Prueba con umbrales de vl_ubcmatch, pico y borde desiguales y porcentaje de puntos clave aceptados

Ante los resultados de las cuatro pruebas anteriores, se siguen obteniendo unas tasas de reconocimiento muy bajas, por lo que se estudia una manera de obtener unos mejores resultados. Si de alguna forma se pudieran obtener unas buenas características para el entrenamiento, posteriormente en la etapa del reconocimiento sería mucho más fácil localizarlas. En la extracción de características durante la búsqueda de coincidencias se deberían obtener sólo las características más importantes para observar las que coinciden verdaderamente con las del logotipo buscado y extraídas durante el entrenamiento (no las que son parecidas con las características de otros logotipos distintos). Si se dieran unos valores para la detección de *picos* y *bordes* más benévolos en la etapa del entrenamiento y posteriormente, en la etapa de extracción de características a la hora del reconocimiento el sistema fuera más severo, habría una mayor posibilidad de conseguir unas características coincidentes mejores y por lo tanto una probabilidad de detección del logotipo mucho mayor.

Por todo esto, manteniendo los niveles de vl_ubcmatch a 1,25 y de obtención de más del 50% de características coincidentes para considerar que existe el logotipo buscado, se modifican los valores de *pico* para realizar una extracción de puntos clave en el entrenamiento con un valor 20 y durante el reconocimiento con un valor 30. Los resultados obtenidos con esta prueba fueron los siguientes.

Tabla 16 - Tabla de Contingencia global (nivel de pico 20 en entrenamiento y 30 en reconocimiento, vl_ubcmatch 1,25 y más del 50% de puntos clave).

TP = 1	FN = 17
FP = 29	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 1/(1+17) = 0,0588 (5,88%)
R = 1/(1+29) = 0,0333 (3,33%)
F1 = (2·0,588·0,0333)/(0,0588+0,0333) = 0,0425 (4,25%)

Manteniendo de nuevo los niveles de vl_ubcmatch a 1,25 y de obtención de más del 50% de características coincidentes para considerar que existe el logotipo buscado, se modifican los valores de *borde* para realizar una extracción de puntos clave en el entrenamiento con un valor 10 y durante el reconocimiento con un valor 5, obteniendo los siguientes resultados.

Tabla 17 - Tabla de Contingencia global (nivel de borde 10 en entrenamiento y 5 en reconocimiento, vl_ubcmatch 1,25 y más del 50% de puntos clave).

TP = 13	FN = 5
FP = 81	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 13/(13+81) = 0,1382 (13,82%)
R = 13/(13+5) = 0,7222 (72,22%)
F1 = (2·0,1382 ·0,7222)/(0,1382+0,7222) = 0,2320 (23,20%)

Si se realiza una media de estas pruebas se obtienen los siguientes valores:
P = 0,0985 (9,85%)
R = 0,3775 (37,75%)
F1 = 0,1372 (13,72%)

Se observan unos resultados bastante decepcionantes, por lo que se analiza otro método para realizar un reconocimiento mucho más eficiente.

5.1.5.- Prueba con umbrales de vl_ubcmatch, porcentaje de puntos clave aceptados y porcentaje de altura necesaria para ser aceptado

Si de alguna forma se consiguiera separar el logotipo que aparece en cada imagen del resto, aunque estos sobrepasen el primer umbral de contener un número de coincidencias superior a un porcentaje de las almacenadas durante el entrenamiento, se podrían obtener unos mejores resultados. Se observa que el logotipo que aparece en el sistema, en el mayor número de casos tiene un número de coincidencias muy superior al resto de los que han pasado el umbral del porcentaje, por esta razón, se realiza un segundo filtrado, aceptando como buenos los logotipos en los que se ha obtenido un número de coincidencias parecido al que ha obtenido el mayor número de ellas.

Este método tiene un inconveniente principal y es que en el caso de tener dos logotipos en la misma imagen de entrada, podría darse el caso de que uno de ellos, aun superando el porcentaje de características coincidentes con las del entrenamiento no llegara al segundo umbral para considerar que existe un logotipo. Para solucionar este problema se realizarían búsquedas en las que los logotipos que se intenten localizar tuvieran una cantidad parecida de características extraídas durante el entrenamiento.

Durante esta última prueba se realizaron muchas evaluaciones con distintos valores para los umbrales impuestos. En la que se obtuvieron unos mejores resultados se utilizó un nivel para la búsqueda (*vl_ubcmatch*) un umbral de valor 1,25, un porcentaje algo menor a los de las pruebas anteriores (50%), para evitar que los logotipos con menos puntos clave localizados se queden fuera de los posibles candidatos a aparecer y se fijó un porcentaje del 90% de la altura del número de coincidencias mayor para que se considere que el logotipo está en la imagen de entrada. Con todos estos parámetros se obtuvieron estos resultados:

Tabla 18 - Tabla de Contingencia global (vl_ubcmatch 1,25, más del 50% de puntos clave y 90% del mayor número de coincidencias).

TP = 11	FN =7
FP = 2	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 11/(11+2) = 0,8461 (84,61%)
R = 11/(11+7) = 0,6111 (61,11%)
F1 = (2·0,8461 ·0,6111)/(0,8461+0,6111) = 0,7096 (70,96%)

Tras observar la gran mejoría en los datos de este tipo de reconocimiento, se decide volver a comprobar el sistema al completo para ver la precisión con los distintos tipos de imágenes de entrada, manteniendo los valores de los umbrales de la última prueba.

- Imágenes individuales

Tabla 19 - Tabla de Contingencia imágenes individuales (vl_ubcmatch 1,25, más del 50% de puntos clave y 90% del mayor número de coincidencias).

TP = 7	FN = 6
FP = 2	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 7/(7+2) = 0,7777 (77,77%)
R = 7/(7+6) = 0,5384 (53,84%)
F1 = (2·0,7777 ·0,5384)/(0,7777+0,5384) = 0,6363 (63,63%)

- Imágenes con cambios de escala.

Tabla 20 - Tabla de Contingencia imágenes con cambios de escala (vl_ubcmatch 1,25, más del 50% de puntos clave y 90% del mayor número de coincidencias).

TP = 7	FN = 4
FP = 1	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 7/(7+1) = 0,8750 (87,50%)
R = 7/(7+4) = 0,6363 (63,63%)
F1 = (2·0,8750 ·0,6363)/(0,8750+0,6363) = 0,7367 (73,67%)

- Imágenes con giros de menos de 30 grados.

Tabla 21 - Tabla de Contingencia imágenes con giros menores de 30 grados (vl_ubcmatch 1,25, más del 50% de puntos clave y 90% del mayor número de coincidencias).

TP = 8	FN = 5
FP = 2	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 8/(8+2) = 0,8 (80%)
R = 8/(8+5) = 0,6153 (61,53%)
F1 = (2·0,8 ·0,6153)/(0,8+0,6153) = 0,7479 (74,79%)

- Imágenes con giros mayores de 30 grados.

Tabla 22 - Tabla de Contingencia imágenes con giros mayores a 30 grados (vl_ubcmatch 1,25, más del 50% de puntos clave y 90% del mayor número de coincidencias).

TP = 7	FN = 5
FP = 8	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 7/(7+8) = 0,4666 (46,66%)
R = 7/(7+5) = 0,5833 (58,33%)
F1 = (2·0,4666 ·0,5833)/(0,4666+0,5833) = 0,5184 (51,84%)

- Imágenes con distinta iluminación.

Tabla 23 - Tabla de Contingencia imágenes con cambios de iluminación (vl_ubcmatch 1,25, más del 50% de puntos clave y 90% del mayor número de coincidencias).

TP = 9	FN = 4
FP = 2	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 9/(9+2) = 0,8181 (81,81%)
R = 9/(9+4) = 0,6923 (69,23%)
F1 = (2·0,8181 ·0,6923)/(0,8181+0,6923) = 0,7499 (74,99%)

- Imágenes ocluidas.

Tabla 24 - Tabla de Contingencia imágenes con oclusiones (vl_ubcmatch 1,25, más del 50% de puntos clave y 90% del mayor número de coincidencias).

TP = 6	FN = 7

FP = 5	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 6/(6+7) = 0,4615 (46,15%)
R = 6/(6+5) = 0,5454 (54,54%)
F1 = (2·0,4615·0,5454)/(0,4615+0,5454) = 0,4999 (49,99%)

- Imágenes múltiples.

Tabla 25 - Tabla de Contingencia imágenes múltiples (vl_ubcmatch 1,25, más del 50% de puntos clave y 90% del mayor número de coincidencias).

TP = 7	FN = 7
FP = 0	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 7/(7+0) = 1 (100%)
R = 7/(7+7) = 0,5 (50%)
F1 = (2·1 ·0,5)/(1+0,5) = 0,6666 (66,66%)

- Imágenes en contexto.

En este caso, se realizaron varias pruebas, puesto que el número de características coincidentes para todas las imágenes almacenadas era muy elevado y el primero de los umbrales (más del 50% de puntos clave coincidentes) era sobrepasado demasiado fácilmente. Los mejores resultados se obtuvieron para los umbrales: (vl_ubcmatch 1,5, más del 80% de puntos clave coincidentes y 90% del mayor número de coincidencias).

Tabla 26 - Tabla de Contingencia imágenes en contexto (vl_ubcmatch 1,5, más del 80% de puntos clave y 90% del mayor número de coincidencias).

TP = 5	FN = 6
FP = 6	---

Los valores de *Precisión, Cobertura* y *F1* fueron los siguientes:
P = 5/(5+6) = 0,4545 (45,45%)
R = 5/(5+6) = 0,4545 (45,45%)
F1 = (2·0,4545 ·0,4545)/(0,4545+0,4545) = 0,4590 (45,90%)

La tabla que resume los resultados para cado una de los tipos de imágenes es la siguiente.

Tabla 27 - Resumen de valores P, R y F1 (vl_ubcmatch 1,5, más del 80% de puntos clave y 90% del mayor número de coincidencias).

TIPO DE IMAGEN	Nombre de la prueba	P	R	F1
Individuales	A	0,7777	0,5384	0,6363
Con cambios de escala	B	0,8750	0,6363	0,7367
Con giros < 30 grados	C	0,8000	0,6153	0,7479
Con giros > 30 grados	D	0,4666	0,5833	0,5184
Con cambios de iluminación	E	0,8181	0,6923	0,7499
Con oclusiones parciales	F	0,4615	0,5454	0,4999
Múltiples	G	1,0000	0,5000	0,6666
En contexto	H	0,4545	0,4545	0,4590
MEDIA		0,7067	0,5707	0,6268

La gráfica asociada a estos resultados es la siguiente.

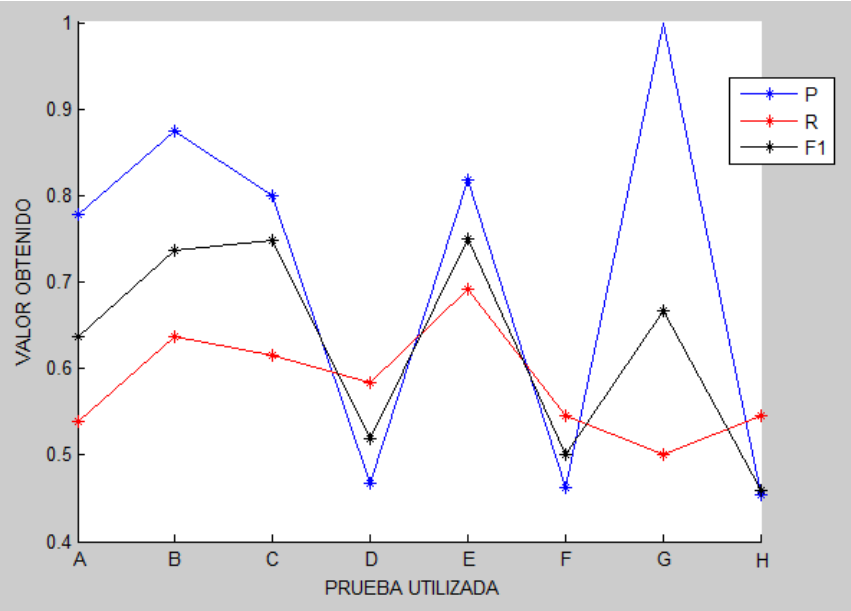


Imagen 33 - Gráfica resumen de valores P, R y F1 (vl_ubcmatch 1,5, más del 80% de puntos clave y 90% del mayor número de coincidencias).

Se puede dilucidar con estos resultados que existe una gran mejoría entre no usar ningún tipo de nivel para considerar que existe un logotipo al encontrarse una gran diferencia respecto al mayor número de coincidencias y sí usarlo. El nivel óptimo que se ha encontrado para estas pruebas ha sido del 90%, aunque en algunos casos no se encuentra el logotipo existente y por tanto el número de *FN* aumenta sin tener que hacerlo, los niveles no son del todo malos.

Sigue existiendo el problema de las imágenes con más de un logotipo, aunque en menor grado que anteriormente, en las que se obtiene sólo el resultado que obtiene mayores niveles de coincidencias. La solución anteriormente descrita es la de localizar dos logotipos con un número parecido de puntos clave, para que a la hora de localizarlos en la imagen obtengan un número parecido de coincidencias y sea posible determinar que los dos están en la imagen de entrada.

Donde peores resultados se obtienen es en las imágenes ocluidas, en las que están en contexto y las que tienen giros de más de 30 grados. Debido a la falta de puntos clave en el primero de los casos y al exceso de los mismos en el segundo se obtiene una precisión muy baja en ambos casos. En el caso de los giros de más de 30 grados los resultados tampoco son satisfactorios, pero en las investigaciones de Lowe ya se mencionaba que podrían existir peores resultados que con los giros menores de 30 grados.

5.2.- CONCLUSIONES.

Se puede observar que algunos tipos de imágenes se reconocen mejor que otros, por lo que puede concluirse que el sistema trabaja mejor para el reconocimiento de algunos logotipos en concreto. Por ejemplo, el sistema tiene una mejor respuesta para imágenes sin ningún tipo de cambio, cambios de escala, de iluminación y giros de menos de 30 grados. De igual forma se puede observar que el sistema no tiene un buen funcionamiento para imágenes en contexto, ocluidas, con giros de más de 30 grados y con varios logotipos (a no ser que todos los que aparezcan en una misma imagen tengan un número parecido de características).

Si se determina un punto de vista más general, se puede decir que el algoritmo para reconocer logotipos trabaja mejor con imágenes complejas, es decir con una cantidad más alta de características, pues de ese modo se hace mucho más sencillo diferenciarlas del resto de objetos almacenados durante el entrenamiento.

Como se ha descrito en el apartado de las pruebas realizadas, es posible ver qué umbrales han sido utilizados finalmente en el sistema.

Los umbrales que podían ser utilizados para el control del número de características encontradas durante la realización de la Transformada SIFT (de *pico* y *borde*) no tienen mucho sentido, puesto que es mejor obtener un gran número de puntos clave para tener más opciones en su búsqueda en otras imágenes; por tanto estos dos umbrales no son necesarios realmente y solo servirían en caso de que no se desearan obtener un gran número de características para cada imagen de entrenamiento (por ejemplo por falta de espacio en el disco duro).

El umbral que determina la distancia euclídea que debe tener un punto respecto a los vecinos más próximos para poder ser elegido como coincidencia está impuesto mediante el parámetro que se cambiará en la función vl_ubcmatch propuesta por VL_FEAT para la búsqueda de coincidencias. El valor elegido para la realización de las pruebas es de 1,25, aunque en algunos casos (como el de logotipos en contexto) el valor será algo mayor (1,5).

Por otra parte, deben coexistir otros dos porcentajes que se deben superar para que se considere que un logotipo aparece en la imagen de entrada. Uno de ellos es el que determina la cantidad de puntos clave coincidentes entre la imagen de entrada y la imagen almacenada durante el entrenamiento; si se supera este primer filtrado, se pasará a un segundo que determinará la diferencia entre el logotipo que tiene más posibilidades de aparecer en la imagen y el resto de ellos. Si en alguna imagen existen dos (o más) logotipos, deberían tener un número parecido de características, puesto que si no solo se considerará el más coincidente como el que aparece en la imagen.

Los valores óptimos para esos dos umbrales son el 50% para el primero de ellos y el 90% para el segundo, aunque en los casos en los que se obtiene un gran número de características en la imagen se debería imponer un valor más alto en el primero.

Tabla 28 - Resumen de los resultados de las pruebas realizadas.

NOMBRE DE LA PRUEBA	PRUEBA REALIZADA	P	R	F1
A	Prueba con filtrado de scores y puntos clave aceptados	0,2741	0,5002	0,3134
B	Prueba con umbral vl_ubcmatch y porcentaje de puntos clave aceptados	0,1223	0,4444	0,1926
C	Prueba con umbral vl_ubcmatch, pico y borde iguales y porcentaje de puntos clave aceptados	0,1178	0,6666	0,1997
D	Prueba con umbral vl_ubcmatch, pico y borde desiguales y porcentaje de puntos clave aceptados	0,0985	0,3775	0,1372
E	Prueba con umbral vl_ubcmatch, porcentaje de puntos clave aceptados y de altura necesaria para ser aceptados	0,7067	0,5707	0,6268

Observando la tabla que resume los resultados de las distintas pruebas, se puede deducir finalmente que a pesar de la dificultad del sistema para localizar fielmente los logotipos existentes en una imagen, se puede observar una localización del 62,68% de fiabilidad del sistema frente al 31,34% obtenido durante la primera de las pruebas, por lo que las investigaciones realizadas para el reconocimiento e identificación de logotipos mediante Transformada SIFT han aportado resultados interesantes y positivos, si bien es verdad que para otros usos de la Transformada se puedan obtener unos niveles de reconocimiento mayores.

La progresión obtenida en los niveles P,R y F1 durante la realización de las distintas pruebas es la que se representa en la siguiente gráfica, donde los nombres de las pruebas se han sustituído por las letras correspondientes, indicadas en la tabla anterior.

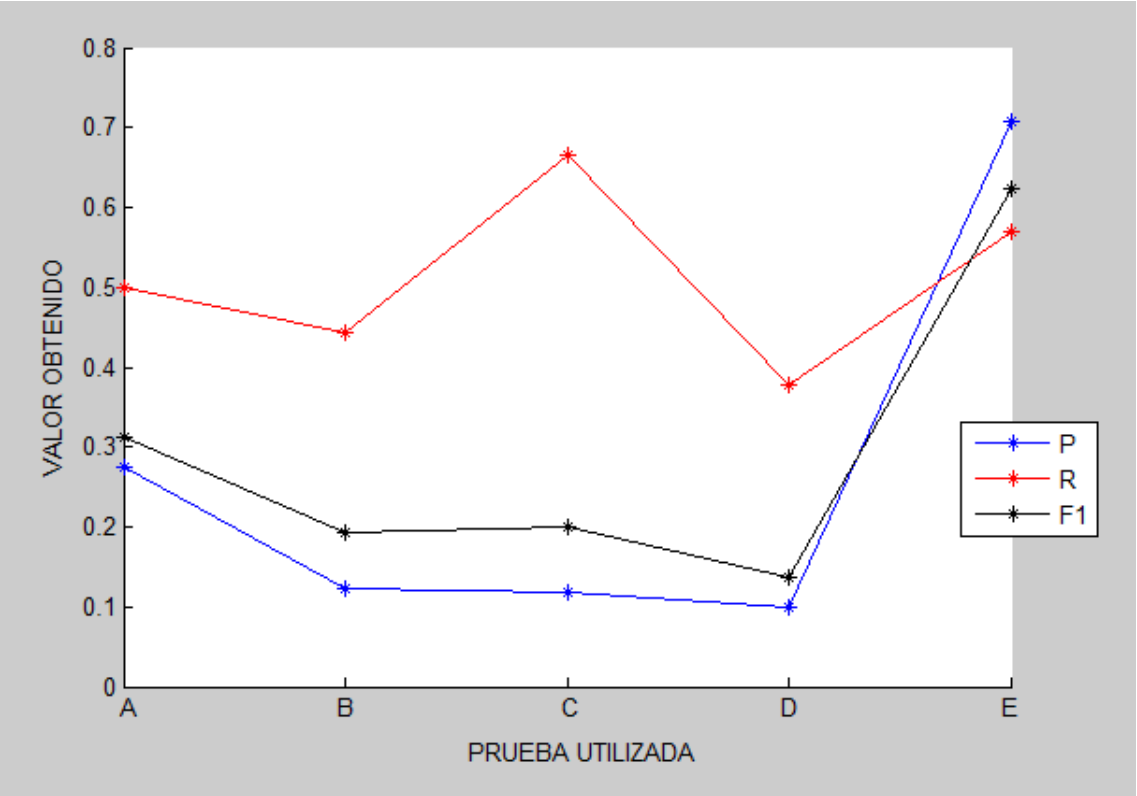


Imagen 34 - Gráfica resumen de resultados P, R y F1 obtenidos en las pruebas.

Como se puede observar, los valores de la última de las pruebas (E) son muy superiores a los realizados durante las cuatro primeras (A, B, C y D) como ya se podía observar en la tabla 28.

6.- MEJORAS Y TRABAJOS FUTUROS

La realización de un reconocimiento automatizado de imágenes es cada vez más una realidad. Con la cantidad de información multimedia lanzada a internet a diario por millones de usuarios, hace que su tratamiento, etiquetado, comprobación, etc, sea más que imposible de realizar de forma manual. Es por esto, por lo que la realización de programas o sistemas de reconocimiento sea una de las líneas de investigación abiertas actualmente y desde hace varios años, y que constantemente está cambiando y generando nuevas líneas de trabajo futuro.

El programa implementado forma parte de una investigación para un Proyecto Fin de Carrera, con lo que no ha sido implementado con un ánimo de lucro, donde se habría puesto mucho más empeño en la compatibilidad (traducción a otros idiomas, escalabilidad, etc), tiempo de procesamiento, resultados obtenidos, etc. Son muchas las mejoras y posibles inclusiones que el programa puede generar tras su finalización y por tanto es necesario mencionar al menos las más importantes.

Entre las posibles mejoras se destaca la adicción de métodos más rápidos y eficientes para la realización del reconocimiento, puesto que no se ha puesto demasiado empeño en realizar un sistema que pueda utilizarse en tiempo real, sino que se ha realizado una investigación sobre la forma de realizar un reconocimiento de forma fiable con las herramientas proporcionadas, que se ha cubierto de forma mucho más que suficiente.

Una de las ideas que se gestó inicialmente como uno de los puntos fuertes del proyecto es la inclusión de un reconocimiento vía webcam, que haría mucho más indispensable las acciones por parte del usuario, y por tanto un uso más ameno. Finalmente, tras mucho deliberar se ha decidido no lanzarse a realizar la aplicación, puesto que no llegaría a aportar ninguna nueva conclusión a los resultados obtenidos en esta investigación.

La inclusión de otras técnicas de reconocimiento de imágenes, combinada con las utilidades de la Transformada SIFT aquí descritas, sería otra de las mejoras que podría hacer que este sistema tuviera una mayor fiabilidad a la hora de identificar los logotipos.

Otra línea interesante para la investigación de la Transformada SIFT sería la prueba con otras bases de datos de imágenes como por ejemplo: portadas de libros, CD´s, DVD´s, carteles de películas de cine, etc.,

Un sistema de reconocimiento de emisión de anuncios de televisión también sería otro gran uso del reconocimiento visual de imágenes, pues un vídeo no es más que una secuencia de imágenes que habría que reconocer para saber si el “spot” ha salido a la hora correcta, en el sitio indicado, etc de forma automática. Este tipo de reconocimiento se debería realizar en tiempo real y las técnicas utilizadas para el reconocimiento de logotipos descritas en este Proyecto Fin de Carrera podrían utilizarse de forma que se observara si el logotipo de una marca o un producto aparece o no en la pantalla.

Otra de las mejoras que se podría implementar es la opción de incorporar o eliminar logotipos del sistema sin necesidad de volver a realizar el entrenamiento del mismo, aunque es una opción que en la investigación que se ha llevado a cabo no es imprescindible.

Como se puede observar, el campo de investigación derivado de este Proyecto Fin de Carrera es muy amplio, ya que el entorno del reconocimiento automático de imágenes está en

constante avance y las mejoras y nuevas técnicas crecen diariamente, por lo que sería un buen terreno para próximos Proyectos y Trabajos de Investigación.

PRESUPUESTO

1.- RECURSOS HUMANOS

La duración total del proyecto ha sido de 7 meses, comprendidos entre marzo y noviembre de 2011. Durante estos siete meses, se ha contado con el trabajo de un Ingeniero de Telecomunicación como jefe de proyecto y de un Ingeniero Técnico de Telecomunicación como programador encargado de realizar la implementación del sistema y la redacción de la memoria del proyecto.

El salario de un Ingeniero es de unos 2.000€/mes, aunque teniendo en cuenta la situación se puede fijar en 1.800€/mes. Si ha realizado un trabajo de 7 meses, el importe por este servicio asciende a 12.600€.

El sueldo en el mercado actual para un Ingeniero Técnico es de unos 1.400€/mes, aunque en la situación actual, se puede establecer ese salario como 1.200€/mes. Por tanto el valor del trabajo de 7 meses es de 8.400€.

En concepto de recursos humanos, se puede entonces afirmar que se ha generado un gasto total de: **21.000 €.**

2.- EQUIPOS

El ordenador personal utilizado para la realización del Proyecto Fin de Carrera es un Acer con un procesador Pentium M a 1,6GHz y Windows XP Home. El uso de este portátil fue desde marzo hasta octubre de 2011. El precio de dicho ordenador fue de 900 €, aunque el uso para la realización del proyecto ha sido sólo durante un 5 meses.

Durante el desarrollo del proyecto se ha adquirido un segundo ordenador personal de marca ASUS con un procesador INTELCore I5 a 2,3GHz con Windows 7. El precio inicial del portátil ha sido de 700 €, y se ha dado un uso a este sistema de 2 meses.

Se calcula el valor de amortización, utilizando la fórmula:

$$\frac{A}{B} \cdot C \cdot D \qquad [23]$$

Donde A es el número de meses desde la fecha de facturación en que el equipo es utilizado, B es el valor de depreciación en meses, este caso 60 meses, C es el coste del equipo (sin IVA) y D es el porcentaje de uso que se dedica al proyecto, en este proyecto será 100%.

El total derivado del uso de los ordenadores personales, teniendo en cuenta la depreciación será de: **98,33€.**

3.- OTROS COSTES DIRECTOS

LICENCIAS DE PROGRAMAS

Se hizo necesaria la utilización de una licencia de Microsoft Office 2007, MATLAB 2008 y Macromedia Fireworks 2003. Teniendo en cuenta, que dichos programas serán utilizados para otros medios una vez finalizado el Proyecto Fin de Carrera, se puede decir que el porcentaje de uso de dichas licencias solo llega al 25% de su uso total. Los gastos derivados del porcentaje de estas licencias son los siguientes:

Microsoft Office 2007: 709 * 0,25 = 177,25€
MATLAB 2008: 500 * 0,25 = 125€
Macromedia Fireworks 2003: 137.78 * 0,25 = 34,5€

La cantidad derivada del uso de las licencias asciende a: 104,5€.

En concepto de costes materiales, se obtiene un gasto total de: **104,5€**

4.- COSTES INDIRECTOS

VIAJES

Cada vez que se ha necesitado acudir a una tutoría, se ha requerido utilizar el transporte público, con lo que el gasto (considerando el descuento realizado por carnet joven y de familia numerosa) ha sido de 6,5 € por viaje.
A razón de 10 tutorías con 2 viajes cada una de ellas, se ha realizado un gasto de 130€.

COSTES DE LUZ

Los costes derivados del gasto de luz y calefacción (eléctrica con tarificación nocturna), durante estos 7 meses de duración del proyecto tienen una media de 50 €, con lo que el gasto total derivado de el uso de la vivienda particular ha sido de 350€.

ADSL

El coste de la conexión ADSL necesaria para la realización del proyecto, supone un gasto de 35€/mes, por lo que el total gastado para este fin es de: 245€.

En concepto de costes indrectos, se obtiene un gasto total de: **595€**

5.- TOTAL

Al coste obtenido, se le debe añadir el correspondiente Impuesto de Valores Agregado (IVA). En este caso, se añadirá un 18%, obteniendo un total de 3.947€.

El coste total del proyecto, incluyendo el impuesto anterior será de **25.847,5€.**

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- AUTOR: César Juárez Megías.

2.- DEPARTAMENTO: Ingeniería Telemática.

3.- DESCRIPCIÓN DEL PROYECTO

- TÍTULO: Reconocimiento e identificación de logotipos en imágenes con Transformada SIFT
- DURACIÓN (meses): 7
- TASAS DE COSTES INDIRECTOS: 18%

4.- PRESUPUESTO TOTAL DEL PROYECTO (valor en €): 25.847,5 €.

5.- DESGLOSE PRESUPUESTARIO:

- RECURSOS HUMANOS

Apellidos y nombre	NIF	Categoría	Dedicación (meses)	Coste mes	Coste (€)
Villena Román, Julio	-	Ingeniero	7	1.800,00	12.600
Juárez Megías, César	-	Ingeniero técnico	7	1.200,00	8.400
TOTAL HORAS			14	**TOTAL**	21.000

- EQUIPOS

Descripción	Coste (€)	% uso dedicado al proyecto	Dedicación (meses)	Período de depreciación	Coste imputable
Ordenador ASUS	700,00	100	2	60	23,33
Ordenador ACER	900,00	100	5	60	75
				TOTAL	98,33

- OTROS COSTES DIRECTOS DEL PROYECTO

Descripción	Empresa	Costes imputables
Licencias de programas	Office, Matlab, Macromedia	104,5
	TOTAL	104,5

- COSTES INDIRECTOS

Descripción	Empresa	Costes imputables
10 viajes a tutorías (ida y vuelta)	RENFE	130
Costes de luz	IBERDROLA	350
Costes ADSL	YA.COM	245
	TOTAL	725

6.- RESUMEN DE COSTES:

Presupuesto Costes Totales	Presupuesto Costes Totales (€)
Personal	21.000
Amortización de equipos	98,33
Otros costes directos	104,5
Costes indirectos	725
IVA	3.947
TOTAL	**25.847,5**

BIBLIOGRAFÍA

[1] Delbracio, M., Aguerrebere, C., Capdehourat, G. y Mateu., M. *"Reconocimiento automático de caras"*
Proyecto fin de carrera. (2006)

[2] Barchiesi, J.V., *"Introducción al procesamiento digital de Señales".*
Ediciones Universitarias de Valparaíso. (2008)

[3] www.MiTecnologico.com, *"Breve Historia De La Graficacion".*
http://www.mitecnologico.com/Main/BreveHistoriaDeLaGraficacion. (Último acceso: Octubre 2011)

[4] Moravec, H. "*Rover visual obstacle avoidance*".
In International Joint Conference on Artificial Intelligence, Vancouver, Canada, pags. 785-790. (1981)

[5] Harris, C. y Stephens, M. "*A combined corner and edge detector*".
In Fourth Alvey Vision Conference, Manchester, UK, pags. 147-151. (1988)

[6] Urbaez, W. *"Técnicas de diseño".*
URL: http://www.desarrolloweb.com/articulos/2183.php (Último acceso: Octubre 2011)

[7] Harris, C*. "Geometry from visual motion".*
In Active Vision, editores A. Blake and A. Yuille, MIT Press, pags. 263-284. (1993)

[8] Zhang, Z., Deriche, R., Faugeras, O., and Luong, Q.T. *"A robust technique for matching two un-calibrated images through the recovery of the unknown epipolar geometry".*
Artificial Intelligence, número 78, pags. 87-119. (1995)

[9] Torr, P. *"Motion Segmentation and Outlier Detection".*
Ph.D. Thesis, Dept. of Engineering Science, University of Oxford, UK. (1995)

[10] Schmid, C., y Mohr, R. *"Local grayvalue invariants for image retrieval".*
IEEE Trans. on Pattern Analysis and Machine Intelligence, volumen 19 (número 5), pags. 530-534. (1997)

[11] Lowe, D.G. *"Object recognition from local scale-invariant features".*
In International Conference on Computer Vision, Corfu, Greece, pags. 1150-1157. (1999)

[12] Crowley, J. L. y Parker, A.C. *"A representation for shape based on peaks and ridges in the difference of low-pass transform".*
IEEE Trans. on Pattern Analysis and Machine Intelligence, volumen 6, número 2, pags. 156-170. (1984)

[13] Shokoufandeh, A., Marsic, I., y Dickinson, S.J. *"View-based object recognition using saliency maps".*
Image and Vision Computing, volumen 17, págs. 445-460. (1999)

[14] Lindeberg, T. *"Detecting salient blob-like image structures and their scales with a scale-space primal sketch: a method for focus-of-attention".*
International Journal of Computer Vision, volumen 11, número 3, págs. 283-318. (1993)

[15] Lindeberg, T. *"Scale-space theory: A basic tool for analysing structures at different scales".*
Journal of Applied Statistics, volumen 21, número2, págs. 224-270. (1994)

[16] Baumberg, A. *"Reliable feature matching across widely separated views".*
In Conference on Computer Vision and Pattern Recognition, Hilton Head, South Carolina, págs. 774-781. (2000)

[17] Tuytelaars, T., y Van Gool, L. *"Wide baseline stereo based on local, affinely invariant regions".*
In British Machine Vision Conference, Bristol, UK, págs. 412-422. (2000)

[18] Mikolajczyk, K., y Schmid, C. *"An affine invariant interest point detector".*
In European Conference on Computer Vision (ECCV), Copenhagen, Denmark, págs. 128-142. (2002)

[19] Schaffalitzky, F., y Zisserman, A. *"Multi-view matching for unordered image sets, or 'How do I organize my holiday snaps?' ".*
In European Conference on Computer Vision, Copenhagen, Denmark, págs. 414-431. (2002)

[20] Brown, M. y Lowe, D.G. *"Invariant features from interest point groups".*
In British Machine Vision Conference, Cardiff, Wales, págs. 656-665. (2002)

[21] Matas, J., Chum, O., Urban, M., y Pajdla, T. *"Robust wide baseline stereo from maximally stable extremal regions".*
In British Machine Vision Conference, Cardiff, Wales, págs. 384-393. (2002)

[22] Mikolajczyk, K., Zisserman, A., y Schmid, C. *"Shape recognition with edge-based features".*
In Proceedings of the British Machine Vision Conference, Norwich, U.K.(2003)

[23] Nelson, R.C., y Selinger, A. "Large-scale tests of a keyed, appearance based 3D object recognition system".
Vision Research, volumen 38 número 15, págs. 2469-2488. (1998)

[24] Pope, A.R., y Lowe, D.G. *"Probabilistic models of appearance for 3-D object recognition".*
International Journal of Computer Vision, volumen 40 número 2, págs. 149-167. (2000)

[25] Carneiro, G., y Jepson, A.D. *"Phase-based local features".*
In European Conference on Computer Vision, Copenhagen, Denmark, págs. 282-296. (2002)

[26] Schiele, B., y Crowley, J.L. *"Recognition without correspondence using multidimensional receptive field histograms".*
International Journal of Computer Vision, volumen 36 número 1, págs. 31-50. (2000)

[27] Basri, R., and Jacobs, D.W. *"Recognition using region correspondences".*
International Journal of Computer Vision, volumen 25 número 2, págs.145-166. (1997)

[28] Lowe, D.G. *"Local feature view clustering for 3D object recognition".*
IEEE Conference on Computer Vision and Pattern Recognition, Kauai, Hawaii, págs. 682-688. (2001)

[29] Lowe, D.G. *"Distinctive Image Features from Scale-Invariant Keypoints"*
Computer Science Department of University of British Columbia, Vancouver, B.C., Canada. (2004)

[30] Lowe, D.G. *"Keypoint detector".*
URL: http://www.cs.ubc.ca/~lowe/keypoints/ (Última modificación: Julio 2010)

[31] Vedaldi, A. y Fulkerson, B. *"VL_FEAT".*
URL: http://www.VL_FEAT.org/index.html (Última modificación: Julio 2011)

[32] Laptev, I. y Lindeberg, T. *"Local descriptors for spatio-temporal recognition".*
Workshop on Spatial Coherence for Visual Motion Analysis, Springer Lecture Notes in Computer Science, volumen 3667. págs. 91–103. (2004)

[33] Lowe, D.G, y Helmer, S., *"Object recognition with many local features"*.
Workshop on Generative Model Based Vision, Washington, D.C. (2004).

[34] Lazebnik, S., Schmid, C., and Ponce, J., *"Semi-Local Affine Parts for Object Recognition".*
Proceedings of the British Machine Vision Conference. (2004)

[35] Ke, Y., and Sukthankar, R., *"PCA-SIFT: A More Distinctive Representation for Local Image Descriptors"*.
Computer Vision and Pattern Recognition. (2004)

[36] Mikolajczyk, K. y Cordelia Schmid, C*. "A performance evaluation of local descriptors".*
IEE Transactions on Pattern Analysis & Machine Intelligence, volumen 27, número 10. (2005)

[37] Bay, H., Tuytelaars, T., Gool, L.V., *"SURF: Speeded Up Robust Features".*
Proceedings of the ninth European Conference on Computer Vision, (2006)

[38] Gordon, I. y Lowe, D. G., *"What and where: 3D object recognition with accurate pose".*
Toward Category-Level Object Recognition, Springer, Verlag, págs. 67-82. (2006)

[39] Laptev, I., Caputo, B., Schuldt, C., y Lindeberg, T., *"Local velocity-adapted motion events for spatio-temporal recognition".*
Computer Vision and Image Understanding, volumen 108, número 3, págs. 207–229 (2007)

[40] Scovanner, P., Ali, S., Shah, M., *"A 3-dimensional sift descriptor and its application to action recognition".*
Proceedings of the 15th International Conference on Multimedia. págs. 357–360. (2007)

[41] Wang, Y., Zhang Bin, Z., Ge, Y., *"The Invariant Relations of 3D to 2D Projection of Point Sets".*
Journal of Pattern Recognition Research, volumen 3, número 1 (2008)

[42] Wagner, D., Reitmayr, G., Mulloni, A., Drummond, T., y Schmalstieg, D., *"Pose tracking from natural features on mobile phones".*
Proceedings of the International Symposium on Mixed and Augmented Reality, (2008)

[43] Cui, Y., Hasler, N., Thormaehlen, T., Seidel, H.-P., *"Scale Invariant Feature Transform with Irregular Orientation Histogram Binning".*
Proceedings of the International Conference on Image Analysis and Recognition, Halifax, Canada: Springer. (2009)

[44] Pardo Feijoo, S., *"Aplicación del modelo BAG-OF-WORDS al reconocimiento de imágenes".*
Proyecto Fin de Carrera. Universidad Carlos III de Madrid. (2009)

[45] Peraza Domínguez, J.M., *"Estimación de la distancia recorrida por un robot móvil mediante la utilización de descriptores SURF".*
Proyecto Fin de Carrera. Universidad Carlos III de Madrid. (2009)

[46] Tarrés González, F., *"Detección y asociación automática de puntos característicos para diferentes aplicaciones".*
Proyecto Fin de Carrera. Universidad Politécnica de Cataluña. (2009)

[47] Toews, M., Wells III, W. M., Collins, D. L. y Arbel, T., *"Feature-based morphometry: discovering group-related anatomical patterns".*
Dans NeuroImage, volumen 49, número 3, págs. 2318-232. (2010)

[48] Flitton, G., Breckon, T., *"Object Recognition using 3D SIFT in Complex CT Volumes".*
Proceedings of the British Machine Vision Conference. págs. 11.1-12. (2010)

[49] Babaud, J., Witkin, A. P., Baudin, M., and Duda, R. O., *"Uniqueness of the gaussian kernel for scale-space filtering".*
IEEE Transactions on Pattern Analysis and Machine Intelligence volumen 8, págs. 26–33. (1986)

[50] Doorn A.J. Van, Koenderink J.J., *"Spatiotemporal integration in the detection of coherent motion".*
Vision Res. volumen24, págs. 47-53. (1984)

[51] Edelman, S., y Duvdevani-Bar, S., *"Similarity-based viewspace interpolation and the categorization of 3D objects".*
Proc. Edinburgh Workshop on Similarity and Categorization, págs. 75-81. (1997)

[52] Friedman, J.H., Bentley, J. L. y Finkel, R. A., *"An algorithm for finding best matches in logarithmic expected time".*
ACMTrans on Mathematic al Software, volumen 3, número 3, págs. 209-226. (1977)

[53] Beis, J. S. y Lowe, D. G., *"Shape indexing using approximate nearest-neighbour search in high-dimensional spaces" .*
Conference on Computer Vision and Pattern Recognition, Puerto Rico, págs. 1000-1006 (1997)

[54] Arya, S., y Mount, D.M. *"Approximate nearest neighbor queries in fixed dimensions".*
InFourth Annual ACM-SIAM Symposium on Discrete Algorithms, págs. 271-280. (1993)

[55] Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., y Wu, A.Y. *"An optimal algorithm for approximate nearest neighbor searching".*
Journal of the ACM,volumen 45, págs. 891-923. (1998)

[56] Hough, P.V.C. *"Method and means for recognizing complex patterns".*
U.S. Patente número 3069654. (1962)

[57] Ballard, D.H. *"Generalizing the Hough transform to detect arbitrary patterns".*
Pattern Recognition, volumen 13, número 2, págs. 111-122. (1981)

[58] Grimson, E. *"Object Recognition by Computer: The Role of Geometric Constraints".*
The MIT Press Cambridge, MA. (1990)

[59] Luong, Q.T., y Faugeras, O.D. *"The fundamental matrix: Theory, algorithms, and stability analysis".*
International Journal of Computer Vision, volumen 17, número1, págs.43-76. (1996)

[60] Hartley, R. and Zisserman, A. *"Multiple view geometry in computer visión".*
Cambridge University Press Cambridge, UK. (2000)

[61] Lowe, D.G. *"Fitting parameterized three-dimensional models to images".*
IEEE Trans. on Pattern Analysis and Machine Intelligence, volumen 13, número 5, págs. 441-450. (1991)

[62] Funt, B.V. y Finlayson, G.D., *"Color constant color indexing".*
IEEE Trans. on Pattern Analysisand Machine Intelligence, volumen 17, número 5, págs. 522-529. (1995)

[63] DAVID CARRETERO DE LA ROCHA, *"Sistema de reconocimiento de partituras".*
P.F.C. Universidad Carlos III de Madrid, Leganés (Junio, 2009)

ANEXO (ARCHIVOS DEL PROGRAMA)

A continuación se añadirán los archivos de Matlab que componen el programa. Se pueden observar en ellos el código necesario para su funcionamiento y en verde los comentarios que harán la lectura y localización de las distintas partes del código mucho más sencilla.

startup.m

```
function startup
    % startup function starts the program.

    % Clear all the variables and text on MATLAB COMMAND WINDOW.
    clear all;
    clc;

    % On the MATLAB COMMAND WINDOW, write the program starts.
    disp ('Starting the program...');
    disp (' ');
    disp ('Load VL FEAT...');

    % Load the vl_feat toolbox and write that system is ready.
    run('vlfeat-0.9.9/toolbox/vl_setup');
    disp (' ');
    disp ('Recognition system Ready');
    disp (' ');

    % Check if the "features.mat" file exist.
    % If it was created, read the features saved in the last program
execution.
    if (exist('features.mat')== 2)
        load features.mat;
    % If it wasn´t created, create the features file.
    else
        %
        storedImages = [];
        save features.mat storedImages;
    end

    % Call the MAIN window.
    main;

end
```

main.m

```
function varargout = main(varargin)
    % main function starts the menu window.

    % Begin initialization code - DO NOT EDIT
    gui_Singleton = 1;
    gui_State = struct('gui_Name',       mfilename, ...
                       'gui_Singleton',  gui_Singleton, ...
                       'gui_OpeningFcn', @main_OpeningFcn, ...
                       'gui_OutputFcn',  @main_OutputFcn, ...
                       'gui_LayoutFcn',  [] , ...
```

```
                   'gui_Callback',   []);
    if nargin && ischar(varargin{1})
        gui_State.gui_Callback = str2func(varargin{1});
    end

    if nargout
        [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
    else
        gui_mainfcn(gui_State, varargin{:});
    end
    % End initialization code - DO NOT EDIT

    % --- Executes just before main is made visible.
function main_OpeningFcn(hObject, eventdata, handles, varargin)
    %
    % hObject    handle to figure
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    % varargin   command line arguments to main (see VARARGIN)

    % Choose default command line output for main
    handles.output = hObject;

    % Update handles structure
    guidata(hObject, handles);

    % Load the logo´s program.
    logo = imread('programData/logoProgram.jpg','jpg');
    logo = uint8(logo);
    ima = image(logo, 'Parent',handles.logoProgram);
    set(handles.logoProgram, 'Visible', 'off','YDir','reverse','XLim',
get(ima,'XData'),'YLim',get(ima,'YData'));

    % Disable the exit button option.
    set(handles.figure1,'CloseRequestFcn',@closeGui);

    % Set a message on the text box.
    set(handles.text,'String','Please, choose an option.');

    % If the storedImages is empty disable the recognizeImage button.
    load features.mat;
    if isempty(storedImages)
        set(handles.recognizeImage, 'Enable', 'off');
        set(handles.recognizeDirectory,'Enable','off');
    end

function varargout = main_OutputFcn(hObject, eventdata, handles)
    % varargout  cell array for returning output args (see VARARGOUT);
    % hObject    handle to figure
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Get default command line output from handles structure
    varargout{1} = [];

    % --- Executes on button press in recognizeImage.
function recognizeImage_Callback(hObject, eventdata, handles)
```

```
    % hObject    handle to recognizeImage (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Close the MAIN window, and call the RECOGNIZE window.
    closereq;
    recognizeImage;

    % --- Executes on button press in recognizeDirectory.
function recognizeDirectory_Callback(hObject, eventdata, handles)
    % hObject    handle to recognizeDirectory (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Disable the buttons and set on the text box a message.
    set(handles.text,'String','Please, choose a folder.');
    set(handles.recognizeImage, 'Enable', 'off');
    set(handles.recognizeDirectory,'Enable','off');
    set(handles.Train_the_system, 'Enable', 'off');
    set(handles.About, 'Enable', 'off');
    set(handles.Exit, 'Enable', 'on');
    pause(1);

    try

    % Ask the user to choose the folder containing the images to
recognize.
    pathRecognize = uigetdir('', 'Choose the folder of the images to
test the system');


        % If the user doesn´t choose a folder, do nothing.
        if (pathRecognize == 0)

            % Enable the buttons.
            set(handles.recognizeImage, 'Enable', 'on');
            set(handles.recognizeDirectory,'Enable','on');
            set(handles.Train_the_system, 'Enable', 'on');
            set(handles.About, 'Enable', 'on');
            set(handles.Exit, 'Enable', 'on');

        % If the user choose a folder, write on MATLAB COMMAND WINDOW.
        else
            clc
            disp(' ')
            disp('Please, wait while we test the system.');
            disp(' ')

            % Set a message on the text box and disable some buttons.
            set(handles.text,'String','The system is recognizing the
images directory. Please wait for the results file.');
            set(handles.Train_the_system, 'Enable', 'off');
            set(handles.About, 'Enable', 'off');
            set(handles.recognizeImage, 'Enable', 'off');
            set(handles.recognizeDirectory,'Enable','off');
            set(handles.Exit, 'Enable', 'on');
            pause(1);
```

```
            % Test the system with the images of the folder.
            recognizeDirectory(pathRecognize);

            % Wait until the user reads the message.
            uiwait(msgbox('The test has been done. See the results on
the results file.'));
            pause(1);

            load features.mat;
            % Try to open the results text document.
            try
                winopen(info_name);
            catch
            disp('We have an error trying to open the results text
file.');
            end


            % Enable some buttons, set on the text box a message and
write on MATLAB COMMAND WINDOW.
            set(handles.Train_the_system, 'Enable', 'on');
            set(handles.About, 'Enable', 'on');
            set(handles.recognizeImage, 'Enable', 'on');
            set(handles.recognizeDirectory,'Enable','on');
            set(handles.Exit, 'Enable', 'on');
            set(handles.text,'String','The recognition of the logos of
the directory has been done, read the results on the file text. Please
push an option to continue.');
            disp(' ')
            disp('Recognition system ready')
            disp(' ')

            % Wait the user choose an option.
            uiwait(gcf);

            end

        catch
            disp ('Error')

            % Enable the buttons.
            set(handles.Train_the_system, 'Enable', 'on');
            set(handles.About, 'Enable', 'on');
            set(handles.Exit, 'Enable', 'on');
            set(handles.text,'String','Please, choose an option.');
            load features.mat
            if (isempty(storedImages))
                set(handles.recognizeImage, 'Enable', 'off');
                set(handles.recognizeDirectory,'Enable','off');
            else
                set(handles.recognizeImage, 'Enable', 'on');
                set(handles.recognizeDirectory,'Enable','on');
            end
            uiwait(gcf);
        end

    % --- Executes on button press in About.
function About_Callback(hObject, eventdata, handles)
```

```
    % hObject    handle to About (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Close the actual window and open the ABOUT window.
    closereq;
    about;


    % --- Executes on button press in Train_the_system.
function Train_the_system_Callback(hObject, eventdata, handles)
    % hObject    handle to Train_the_system (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    % Load features.mat.
        load features.mat;

        % See if the system has been training before.

        % If the system hasn´t been training before, tell the user we
going to
        % train.
        if (isempty(storedImages))
            uiwait(msgbox('The system is going to train. Please
wait.'));
            pause(1);

        % If the system has been trained to ask the user whether to
continue, removing the logos stored.
        else

            choice = questdlg('If you train the system, all the logos
you added will be deleted. Do you want to
continue?','Warning','Yes','No','Yes');

            switch choice
                % If the user choose "YES" continues the program (do
nothing).
                case 'Yes'

                % If the user choose "NO" or "close the window" return
to menu.
                case 'No'
                    return;
                case ''
                    return;
            end


        end

        % Set on the text box a message and disable some buttons.
        set(handles.text,'String','Please, wait while we train the
system. Look at the progress on MATLAB COMMAND WINDOW');
        set(handles.Train_the_system, 'Enable', 'off');
        set(handles.recognizeImage, 'Enable', 'off');
        set(handles.recognizeDirectory,'Enable','off');
```

```
        set(handles.About, 'Enable', 'off');
        set(handles.Exit, 'Enable', 'off');
        pause(1);

        % Try to read the address of the train images.
        try
            % Ask the user the address of the image folder to train
the system.
            pathTrain = uigetdir('', 'Choose the folder of the images
to train the system');

            % If the user doesn´t give an address, do nothing.
            if (pathTrain == 0)
                % Set a message on the text box and enable some
buttons.
                set(handles.text,'String','Please, choose a correct
folder to train the system.');
                set(handles.Train_the_system, 'Enable', 'on');
                set(handles.About, 'Enable', 'on');
                set(handles.Exit, 'Enable', 'on');

                % If the storedImages isn´t empty enable recognize
buttons.
                if (isempty(storedImages)== 0)
                    set(handles.recognizeImage, 'Enable', 'on');
                    set(handles.recognizeDirectory,'Enable','on');

                end
            % If the user give an address.
            else

                % Write on MATLAB COMMAND WINDOW.
                disp('Please, wait while we train the system.');

                % Train the system with the images of the folder,
savint the training images.
                [storedImages] = train(pathTrain);

                % If the training folder didn´t contain any valid
image.
                if isempty(storedImages)
                    % Wait the user read an error message and set a
message on the text box.
                    uiwait(msgbox('The directory hasn´t got any
suported image to train the system. Please, push the train button
again and select another directory.'));
                    set(handles.text,'String','Please, push again the
training button.');
                    pause(1);
                % If the training folder contained some valid image.
                else
                    % Save again the features file.
                    save features.mat storedImages;

                    % Wait the user read a message window.
                    uiwait(msgbox('The system training has been done
corretcly.'));
                    disp('The system training has been done
corretcly.');
```

```
                    disp('We use the images on the address to train
the system: ');
                    disp(pathTrain);
                    pause(1);

                    % Enable the recognize buttons.
                    set(handles.recognizeImage, 'Enable', 'on');
                    set(handles.recognizeDirectory,'Enable','on');

                end
                % Write a message on the text box.
                set(handles.text,'String','Please, choose an
option.');

                % Enable the buttons and wait the user push a button.
                set(handles.Train_the_system, 'Enable', 'on');
                set(handles.About, 'Enable', 'on');
                set(handles.Exit, 'Enable', 'on');
            end

        % If while we try to train the system, we have an error, catch
it.
        catch

            % Enable some buttons and set text on the text box and on
Matlab command window.
            set(handles.Train_the_system, 'Enable', 'on');
            set(handles.About, 'Enable', 'on');
            set(handles.Exit, 'Enable', 'on');
            set(handles.text,'String','The system has had a mistake
during training. Please, choose an option.');
            clc;
            disp('The system has made ??a mistake during training.
Please, choose an option on the menu window.');
            % Check if we have images stored in the system.
            load features.mat

            % If we haven´t got images, disable the recognize buttons.
            if isempty(storedImages)
                set(handles.recognizeImage, 'Enable', 'off');
                set(handles.recognizeDirectory,'Enable','off');
            % If we have got images, enable the recognize buttons.
            else
                set(handles.recognizeImage, 'Enable', 'on');
                set(handles.recognizeDirectory,'Enable','on');
            end

        end

    % --- Executes on button press in Exit.
function Exit_Callback(hObject, eventdata, handles)
    % hObject    handle to Exit (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Close the actual window and write a message on MATLAB COMMAND
WINDOW.
    closereq;
    clc
```

```
    disp(' ')
    disp ('Closing the system...');
    disp(' ')
    disp('Thank you for use the program.')
    disp(' ')
```

train.m

```
function train (pathTrain)
    % function = train (pathTrain)
    % The path train is read and all image files are processed.
    %
    % StoredImages save three vectors:
    % logoNames is the vector of names of the processed images.
    % keypoints is the keypoints vector´s SITF for each image.
    % descriptors is the vector of descriptor´s SIFT for each image.
    %

    % Define the supported image formats.
    supportedImage = {'.BMP'; '.GIF'; '.HDF'; '.JPEG'; '.JPG'; '.PCX';
'.PNG'; '.TIFF'; '.XWD'; '.bmp'; '.gif'; '.hdf'; '.jpeg'; '.jpg';
'.pcx'; '.png';'.tiff'; '.xwd'};

    % Read that folder contents.
    contentPathTrain = dir(pathTrain);
    numImage = 0;
    storedImages = [];
    haveError = 0;

    % See the folder and see the files.
    for i = 1:length(contentPathTrain(:,1))
        % Write the percentage completed of processed images.
        clc;
        disp(' ');
        percentage = 100*(i/(length(contentPathTrain(:,1))));
        disp(['Training: ' num2str(percentage) ' % completed.']);

        % See that this file is not a directory.
        if (contentPathTrain(i,1).isdir == 0)
            % Read the name of each file.
            fileName = contentPathTrain(i,1).name;
            [pathstr, name, extension, version] = fileparts(fileName);

            acceptedImage = 0;

            % See if the extension is a supported type.
            for j = 1:size(supportedImage,1)
                if strcmp(extension,supportedImage(j,1))
                    acceptedImage = 1;
                    break;
                end
            end

            file = fullfile(pathTrain, fileName);
```

```
            % If the image is aceppted, we processed it.
            if acceptedImage
                % Try to read and do the sift transform to the image.
                try
                    % Read the image.
                    readImage = imread(file);
                    numImage = numImage+1;

                    % Store the name in a vector.
                    storedImages(numImage).name2 = name;

                    % Comprobate or transform the image to grayscale.
                    if size(readImage,3) == 3
                        readImage = single(rgb2gray(readImage));
                    else
                        readImage = single(readImage);
                    end

                    % Do the SIFT Transform and store the "keypoints"
and "descriptors".
                    [keypoin, descripto] = vl_sift(readImage) ;

                    % If we use the edge or peak thresh the lines
lines to be used are as follows (where "peak_thresh_value" and
"edge_thresh_value" are the values of thethresholds we impose.

%                     [keypoin, descripto] = vl_sift(readImage,
'PeakThresh', peak_thresh_value));
%                     [keypoin, descripto] = vl_sift(readImage,
'EdgeThresh', edge_thresh_value)) );

                    % Store the size of the image.
                    storedImages(numImage).sizeX = size(readImage,1);
                    storedImages(numImage).sizeY = size(readImage,2);

                    storedImages(numImage).keypoint = keypoin;
                    % Normalize the descriptors [0,1].
                    descripto = double(descripto);
                    descripto = (descripto/255);

                    storedImages(numImage).descriptor = descripto;
                    storedImages(numImage).sizeX = size(readImage,1);
                    storedImages(numImage).sizeY = size(readImage,2);

                % If the read or the sift transform to the image have
an eror, write on the MATLAB COMMAND WINDOW the error and create a
mensaje box.
                catch
                    disp('We have an error reading or doing the sift
transform for a image.');
                    uiwait(msgbox('We have an error reading or doing
the sift transform for a image.'));
                    haveError = 1;
                    pause(1);
                end

            end
        end
    end
```

```
    if (haveError == 0)
        % Write on the MATLAB COMMAND WINDOW that the training has
been completed.
        clc;
        disp(' ');
        disp('The training of the system has been completed.');
        disp (' ');
        disp ('Recognition system Ready');
        disp (' ');
    end

end
```

recognizeImage.m

```
function varargout = recognizeImage(varargin)

    % Begin initialization code - DO NOT EDIT
    gui_Singleton = 1;
    gui_State = struct('gui_Name',       mfilename, ...
                       'gui_Singleton',  gui_Singleton, ...
                       'gui_OpeningFcn', @recognizeImage_OpeningFcn, ...
                       'gui_OutputFcn',  @recognizeImage_OutputFcn, ...
                       'gui_LayoutFcn',  [] , ...
                       'gui_Callback',   []);
    if nargin && ischar(varargin{1})
        gui_State.gui_Callback = str2func(varargin{1});
    end

    if nargout
        [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
    else
        gui_mainfcn(gui_State, varargin{:});
    end
    % End initialization code - DO NOT EDIT

    % --- Executes just before recognizeImage is made visible.
function recognizeImage_OpeningFcn(hObject, eventdata, handles,
varargin)
    % hObject    handle to figure
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    % varargin   command line arguments to recognizeImage (see
VARARGIN)

    % Choose default command line output for recognizeImage
    handles.output = hObject;

    % Update handles structure
    guidata(hObject, handles);

    % Disable the exit button option.
    set(handles.figure1,'CloseRequestFcn',@closeGui);

    % Remove the axis and disable some buttons.
    set(handles.image,'XTick',[],'YTick',[]);
```

```
    set(handles.recognizze,'Enable','off');
    set(handles.setNewLogo,'Enable','off');

    % --- Outputs from this function are returned to the command line.
function varargout = recognizeImage_OutputFcn(hObject, eventdata,
handles)
    % varargout  cell array for returning output args (see VARARGOUT);
    % hObject    handle to figure
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Get default command line output from handles structure
    varargout{1} = [];

     % --- Executes on button press in loadImage.
function loadImage_Callback(hObject, eventdata, handles)
    % hObject    handle to loadImage (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Clear and write on MATLAB COMMAND WINDOW and on the text box.
    clc
    disp(' ')
    disp('The system is working, please wait.')
    set(handles.text,'String','The system is working, please wait.');

    % Load the features file.
    load features.mat;

     % Try to get and show the image.
    try

        % Ask the user the address of the image to recognize.
        [fileName,pathName] = uigetfile('*.*','Choose an image to
recognize');
        % Disable some buttons.
        set(handles.back, 'Enable', 'off');
        set(handles.loadImage, 'Enable', 'off');
        set(handles.recognizze,'Enable','off');
        set(handles.setNewLogo,'Enable','off');
        pause(1);

        % If the user doesn´t give us a correct address enable some
buttons.
        if isequal(fileName,0)
            set(handles.back, 'Enable', 'on');
            set(handles.loadImage, 'Enable', 'on');

        % If the user give us a correct address.
        else
            % Show the image on the window.
            handles.myImage = imread(fullfile(pathName, fileName));
            readImage = handles.myImage;
            imshow(readImage)

            % Write on MATLAB COMMAND WINDOW and save the features.
            clc
```

```
            disp(' ')
            disp('Regognition system ready.')
            save features.mat fileName pathName storedImages;

            % Enable some buttons, disable the "Set new logo" button
and write a message on the text box.
            set(handles.back, 'Enable', 'on');
            set(handles.recognizze, 'Enable', 'on');
            set(handles.loadImage, 'Enable', 'on');
            set(handles.setNewLogo,'Enable','off');
            set(handles.text,'String','Please, push the recognize
button or choose another image.');
        end

    % If we have an error, catch it.
    catch
        % Wait the user read an error message and enable or disable
some buttons.
        uiwait(msgbox('The image can´t be load.'));
        set(handles.back, 'Enable', 'on');
        set(handles.loadImage, 'Enable', 'on');
        set(handles.setNewLogo, 'Enable', 'off');
        set(handles.recognizze, 'Enable', 'off');
        pause(1);

        % Write a text on the MATLAB command window and in the text
box.
        clc;
        disp(' ');
        disp('We have an error while we loading the image. Please
choose another image.');
        set(handles.text,'String','Please, push the recognize button
or choose another image.');
    end

    % Wait the user push any button.
    uiwait(gcf);


    % --- Executes on button press in recognizze.
function recognizze_Callback(hObject, eventdata, handles)
    % hObject    handle to recognizze (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Clear and write on MATLAB COMMAND WINDOW.
    clc;
    disp(' ');
    disp('The system is working, please wait.');

    % Set a tex on the text box and disable some buttons.
    set(handles.text,'String','Please, wait while we recognize the
image.');
    set(handles.recognizze, 'Enable', 'off');
    set(handles.setNewLogo, 'Enable', 'off');
    set(handles.loadImage, 'Enable', 'off');
    pause(1);

    % Try to recognize the image.
```

```
    try
        % Call to recognize.
        recognize;
        % Set a text on the MATLAB Command Window.
        clc;
        disp('The recognition has been made');

        % Load features.mat again.
        load features.mat;

        % Hold on to show the results.
        hold on

        % If we have got results.
        if (exist('results'))

            % Paint a square arround the result.
            for t=1: size (results,2)

                if(results(t).existsLogo)
                    x = [results(t).xMin results(t).xMin
results(t).xMax results(t).xMax results(t).xMin]; % ordinate vector

                    y = [results(t).yMin results(t).yMax
results(t).yMax results(t).yMin results(t).yMin]; % abscissas vector

                    text (results(t).xMax, results(t).yMax,
results(t).nameResult);
                    plot(x, y)
                end
            end

            % Set a text on the text box.
            set(handles.text,'String','The recognition has been made,
please push "Load image" or "Set new logo" button.');

        % If we haven´t got results.
        else
            % Set a text on the text box.
            set(handles.text,'String','There aren´t any logo at the
load image, please push "Load image" or "Set new logo" button.');
        end

        % Hold off to show the results
        hold off

        % Enable or disable some buttons.
        set(handles.recognizze, 'Enable', 'off');
        set(handles.loadImage, 'Enable', 'on');
        set(handles.back, 'Enable', 'on');
        set(handles.setNewLogo, 'Enable', 'on');

        % Clear and wrhite on the MATLAB COMMAND WINDOW.
        clc
        disp(' ')
        disp('Recognition system ready.')

   % If we have an error, we catch it.
    catch
        % Wait the user read a message error.
```

```
        uiwait(msgbox('We can´t recognize the logos of this image'));
        % Set a message on the text box and enable or disable some
buttons.
        set(handles.text,'String','Please, push "Load image" or "Set
new logo" button.');
        set(handles.recognizze, 'Enable', 'off');
        set(handles.setNewLogo, 'Enable', 'on');
        set(handles.loadImage, 'Enable', 'on');
        set(handles.back, 'Enable', 'on');
    end


    % --- Executes on button press in setNewLogo.
function setNewLogo_Callback(hObject, eventdata, handles)
    % hObject    handle to setNewLogo (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Load the features file.
    load features.mat;

    % Ask the user if want to save a logo in the system.
    choice = questdlg('The logo will be saved in the system. Do you
want to continue?','Warning','Yes','No','Yes');

    % Clear and write on the MATLAB COMMAND WINDOW.
    clc
    disp(' ')
    disp('The system is working, please wait.')
    disp(' ')
    pause(1);

    % Comprobate the answer of the user.
    switch choice
        % If choose "YES", set a message on the text box and disable
some buttons.
        case 'Yes'

        set(handles.text,'String','Please, select the logo area and do
double-click over it.');
        set(handles.setNewLogo, 'Enable', 'off');
        set(handles.loadImage, 'Enable', 'off');
        set(handles.recognizze, 'Enable', 'off');
        set(handles.back, 'Enable', 'off');

        % Save the selected area and show it on the window.
        selectedArea = imcrop;
        imshow(selectedArea);

            % If the selected area is RGB, convert it to grayscale.
            if size(selectedArea,3) == 3
                selectedArea = single(rgb2gray(selectedArea));
            else
                selectedArea = single(selectedArea);
            end

            % Set a message on the text box.
            set(handles.text,'String','Please, wait while add the logo
at system.');
```

```
            % Try to get a name for the logo.
            try
                % Ask the user a name and save it.
                nameLogo = inputdlg('Write the logo name: ','Write the
logo name');
                nameLogo = strcat(nameLogo);

                % If the name isn´t correct.
                if (size(nameLogo{1,1},2) == 0)
                    % Wait the user read a message.
                    uiwait(msgBox('Please try again to add the logo at
the system and set a name for save it.'));

                    % Set text on the text box and enable some buttons
                    set(handles.text,'String','The selected logo can´t
been added at system.');
                    set(handles.setNewLogo, 'Enable', 'on');
                    set(handles.loadImage, 'Enable', 'on');
                    set(handles.recognizze, 'Enable', 'on');
                    set(handles.back, 'Enable', 'on');
                    pause(1);

                    % Show the full image again and wait any action.
                    handles.myImage = imread(fullfile(pathName,
fileName));
                    readImage = handles.myImage;
                    imshow(readImage)
                    return;

                % If the name is correct.
                else

                    numImages = size (storedImages,2);


                    % Comprobates if the name has been used for
another logo.
                    for in = 1:numImages

                        % If the name has been used, wait the user
read a message, set a tex ton the text box, enable some buttons and
whor the full image again.
                        if strcmp((nameLogo{1,1}),
(storedImages(in).name2))
                            uiwait(msgBox('The name you want to give
this logo has been ussed.'));
                            set(handles.text,'String','The selected
logo can´t been added at system.');
                            set(handles.setNewLogo, 'Enable', 'on');
                            set(handles.loadImage, 'Enable', 'on');
                            set(handles.back, 'Enable', 'on');
                            set(handles.recognizze, 'Enable', 'on');
                            pause(1);
                            handles.myImage =
imread(fullfile(pathName, fileName));
                            readImage = handles.myImage;
                            imshow(readImage)
                            return;
                        end
```

```
                    end

                    % If the name hasn´t been used, save the
keypoints, the name and the descriptors.
                    numImages = numImages+1;
                    storedImages(numImages).name2 = nameLogo{1,1};

                    [keypoin,descripto] = vl_sift(selectedArea);

                    storedImages(numImages).keypoint = keypoin;
                    descripto = double(descripto);
                    descripto = (descripto/255);

                    storedImages(numImages).keypoint = keypoin;
                    storedImages(numImages).descriptor = descripto;

                    storedImages(numImages).sizeX =
size(selectedArea,1);
                    storedImages(numImages).sizeY =
size(selectedArea,2);

                    save features.mat fileName pathName results
storedImages;

                    % Wait the user read a message, write a text on
the text box and enable some buttons. Wait the user push any button.
                    uiwait(msgbox('The logo has been saved.'));
                    set(handles.text,'String','The selected logo has
been added at system.');
                    set(handles.loadImage, 'Enable', 'on');
                    set(handles.back, 'Enable', 'on');
                    pause(1)
                    uiwait;
                end
            % If we have an error, wait the user read the message
window and wait.
            catch
                msgbox('Error while adding the logo at system. Please
try again.');
                set(handles.loadImage, 'Enable', 'on');
                set(handles.back, 'Enable', 'on');
                pause(1);
                return;
            end

        % If the user push "NO", enable some buttons and wait.
        case 'No'

            set(handles.setNewLogo, 'Enable', 'on');
            set(handles.loadImage, 'Enable', 'on');
            set(handles.back, 'Enable', 'on');
            return;

        % If the user close the window, enable some buttons and wait.
        case ''

            set(handles.setNewLogo, 'Enable', 'on');
            set(handles.loadImage, 'Enable', 'on');
            set(handles.back, 'Enable', 'on');
            return;
```

```
    end

    % Clear and write on the MATLAB COMMAND WINDOW.
    clc
    disp(' ')
    disp('Recognition system ready.')

    % Save the features file.
    save features.mat fileName pathName results storedImages;



    % --- Executes on button press in back.
function back_Callback(hObject, eventdata, handles)
    % hObject    handle to back (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Close the actual window and call the MAIN window.
    closereq;
    main;
```

recognizeDirectory.m

```
function recognizeDirectory (pathRecognize)

    % Load the features file.
    load features.mat;

    % Define the upported image formats.
    supportedImage = {'.BMP'; '.GIF'; '.HDF'; '.JPEG'; '.JPG'; '.PCX';
'.PNG'; '.TIFF'; '.XWD'; '.bmp'; '.gif'; '.hdf'; '.jpeg'; '.jpg';
'.pcx'; '.png';'.tiff'; '.xwd'};

    % Read that folder contents and initialize the variable "images".
    contentPathRecognize = dir(pathRecognize);
    images = 0;

    % Create the Date and Time variables.
    date = clock;
    info_date = ['This document has been created the '
int2str(date(3)) '/' int2str(date(2)) '/' int2str(date(1)) ' at '
int2str(date(4)) ':' int2str(date(5)) ':' int2str(date(6))];
    info_name = ['RESULTS_TEXT_date_' int2str(date(3)) '.'
int2str(date(2)) '.' int2str(date(1)) '_hour_' int2str(date(4)) '.'
int2str(date(5)) '.' int2str(date(6)) '.txt'];

    % Create the text document and write the first line.
    text_document = fopen(info_name, 'wt');
    fprintf(text_document,'%s',info_date);
    fprintf(text_document,'\n');
    fprintf(text_document, '%s',pathRecognize');
    fprintf(text_document,'\n');

    % Try to do the recognition of the images.
    try
```

```
    % See the folder and read the files.
    for e = 1:length(contentPathRecognize(:,1))

        % See that this file is not a directory.
        if (contentPathRecognize(e,1).isdir == 0)

            % Read the name of the file and define the path.
            fileName = contentPathRecognize(e,1).name;
            pathName = pathRecognize;

            % Save the features file.
            save features.mat fileName pathName storedImages;

            % See if the extension is a supported image type.
            acceptedImage = 0;
            [pathstr, name, extension, version] = fileparts(fileName);
            for j = 1:size(supportedImage,1)
                if strcmp(extension,supportedImage(j,1))
                    acceptedImage = 1;
                    break;
                end
            end


            % If the image is aceppted, we processed it.
            if acceptedImage
                images = images+1;

                % Do the recognition.
                recognize;

                % Read the features file, because the results are
within.
                load features.mat;

                % Initialize the variables.
                numberLogo = 0;
                firstLineText = 0;
                existAnyLogo = 0;
                disp(' ');
                nameLogosExist=[];

                % Through all the features of the images.
                for er = 1:size(results,2)

                    % If exist any logo for this result.
                    if (results(er).existsLogo)
                        existAnyLogo = 1;

                        % If the image contains more than a logo,
write the first sentence only once on the results text file.
                        if (firstLineText == 0)
                            disp('')
                            fprintf(text_document,'\n');
                            text = ['The image # ' int2str(images) '
of the directory, with name: ' name ' contens the following logos: '];
                            fprintf(text_document,'%s',text);
                            fprintf(text_document,'\n');
                            firstLineText = 1;
```

```
                        end

                        % Write the name of the logos that appear in
the image on the text file.
                        numberLogo = numberLogo+1;
                        nameLogosExist(images,numberLogo).name =
results(er).nameResult;

fprintf(text_document,'%s',nameLogosExist(images,numberLogo).name);
                        fprintf(text_document,' ');
                        fprintf(text_document,'\n');

                    end

                end

                % If doesn´t exist any logo, write on the text file.
                if (existAnyLogo == 0)
                    fprintf(text_document,'\n');
                    text = ['The image # ' int2str(images) ' of the
directory, with name: ' name ' doesn´t contens any logo.'];
                    fprintf(text_document,'%s',text);
                    fprintf(text_document,'\n');
                end
            end
        end
    end

    % If there aren´t any images, tell the user on MATLAB COMMAND
DISPLAY and on text file this.
    if (images == 0)
        text = ['The introduced directory doesn´t contain any valid
image to test the program.'];
        disp(text)
        fprintf(text_document,'%s',text);
    end

    % If the program has a bug, write on the MATLAB COMMAND DISPLAY
that we have a error from recognizeDirectory.
    catch
      disp('Error from recognizeDirectory');
      % Save the features file.
      save features.mat fileName pathName storedImages nameLogosExist;

      % Close the text document.
      fclose(text_document);
      return;
    end

    % Save the features file.
    save features.mat fileName pathName storedImages nameLogosExist
info_name;

    % Close the text document.
    fclose(text_document);

    % Write on MATLAB COMMAND DISPLAY that the recognition has been
made.
    clc;
```

```
    disp(' ')
    disp('The recognition is done.')
```

recognize.m

```
function recognize

    % Load the features file.
    load features.mat;

    % Read the image to recognize.
    readImage = imread(fullfile(pathName, fileName));

    % Comprobe if is rgb or gray.
    if size(readImage,3) == 3
        readImage = single(rgb2gray(readImage));
    else
        readImage = single(readImage);
    end

    % Do the sift transform.
    [keypoints, descriptors] = vl_sift(readImage) ;

    % If we use the edge or peak thresh the lines lines to be used are
as follows (where "peak_thresh_value" and "edge_thresh_value" are the
values of thethresholds we impose.
    % [keypoints, descriptors] = vl_sift(readImage, 'PeakThresh',
peak_thresh_value));
    % [keypoints, descriptors] = vl_sift(readImage, 'EdgeThresh',
edge_thresh_value)) );


    % Normalize the descriptors [0,1].
    descriptors = double(descriptors);
    descriptors = (descriptors/255);

    % See the numbers of images stored on the training.
    numStoredImages = size(storedImages,2);

    % Initialize the variable "results".
    results = [];

    % Look for similarities between the image and the stored
    % characteristics and save the results.
    for k = 1:numStoredImages
        clc;
        percentage = 100*(k/numStoredImages);
        disp(['Matching ' fileName ':' num2str(percentage) ' %
completed.']);

        [results(k).matchesAccepted, results(k).scoresAccepted] =
vl_ubcmatch(descriptors, storedImages(k).descriptor,1.25);

        numScores(k) = size(results(k).scoresAccepted,2);
        numStoredDescriptors(k) = size(storedImages(k).descriptor,2);
```

```
    % If we use a threshold to accept the matches and scores will use
the following lines.

    % [results(k).matches, results(k).scores] = vl_ubcmatch
(descriptors, storedImages(k).descriptor, 1);

    % numScores = length(results(k).scores);
    % acceptedResults = 0;

    % for n = 1:numScores
    %     if (results(k).scores(n)<1)
    %         acceptedResults = acceptedResults+1;
    %         results(k).matchesAccepted(:,acceptedResults) =
results(k).matches(:,n);
    %         results(k).keypointsX(1,acceptedResults)=
keypoints(1,(results(k).matchesAccepted(1,acceptedResults)));
    %         results(k).keypointsY(1,acceptedResults)=
keypoints(2,(results(k).matchesAccepted(1,acceptedResults)));
    %         results(k).scoresAccepted(acceptedResults) =
results(k).scores(n);
    %     end
    % end

    end

    for m = 1:numStoredImages

        clc;
        percentage = 100*(k/numStoredImages);
        disp(['Recognizing ' fileName ':' num2str(percentage) ' %
completed.']);

        % If the percentage of accepted results is greater than
threshold of the training descriptors.

        if (numScores(m)>(0.5*numStoredDescriptors(m)))


            % If the number of matches is greater than 75% the size of
the image with the highest number of matches, we believe there logo.
            if (numScores(m)>(max(numScores(:))*0.9))
                % Write on a variable that exist a logo and save the
                % position of this.
                keypointsX =
keypoints(1,(results(m).matchesAccepted(1,:)));
                keypointsY =
keypoints(2,(results(m).matchesAccepted(1,:)));
                results(m).existsLogo = 1;
                results(m).xMin = min(keypointsX);
                results(m).yMin = min(keypointsY);
                results(m).xMax = max(keypointsX);
                results(m).yMax = max(keypointsY);
                results(m).nameResult = storedImages(m).name2;
            else
                % Write on a variable that doesn´t exist a logo and
save all variables with value 0.
                results(m).existsLogo = 0;
                results(m).xMin = [];
                results(m).yMin =  [];
                results(m).xMax =  [];
```

```
                results(m).yMax =  [];
                results(m).nameResult =  '';
            end
        else
            % Write on a variable that doesn´t exist a logo and save
            % all variables with value 0.
            results(m).existsLogo = 0;
            results(m).xMin = [];
            results(m).yMin =  [];
            results(m).xMax =  [];
            results(m).yMax =  [];
            results(m).nameResult =  '';
        end
    end

    % Save the features file.
    save features.mat fileName pathName storedImages results;

end
```

about.m

```
function varargout = about(varargin)

    % Begin initialization code - DO NOT EDIT
    gui_Singleton = 1;
    gui_State = struct('gui_Name',       mfilename, ...
                       'gui_Singleton',  gui_Singleton, ...
                       'gui_OpeningFcn', @about_OpeningFcn, ...
                       'gui_OutputFcn',  @about_OutputFcn, ...
                       'gui_LayoutFcn',  [] , ...
                       'gui_Callback',   []);
    if nargin && ischar(varargin{1})
        gui_State.gui_Callback = str2func(varargin{1});
    end

    if nargout
        [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
    else
        gui_mainfcn(gui_State, varargin{:});
    end
    % End initialization code - DO NOT EDIT


    % --- Executes just before about is made visible.
function about_OpeningFcn(hObject, eventdata, handles, varargin)
    % This function has no output args, see OutputFcn.
    % hObject    handle to figure
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    % varargin   command line arguments to about (see VARARGIN)


    % Choose default command line output for about
    handles.output = hObject;

    % Update handles structure
```

```
    guidata(hObject, handles);

    % Disable the exit button option.
    set(handles.figure1,'CloseRequestFcn',@closeGui);


    % --- Outputs from this function are returned to the command line.
function varargout = about_OutputFcn(hObject, eventdata, handles)
    % varargout  cell array for returning output args (see VARARGOUT);
    % hObject    handle to figure
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Get default command line output from handles structure.
    varargout{1} = [];


    % --- Executes on button press in back.
function back_Callback(hObject, eventdata, handles)
    % hObject    handle to back (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % If the button "Back" is pressed, close the actual window, and
open the MAIN window.
    closereq;
    main;


    % --- Executes on button press in openTextFile.
function openTextFile_Callback(hObject, eventdata, handles)
    % hObject    handle to openTextFile (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Try to open the instructions file.
    try
        winopen('programData/instructions.txt');
    % If when you open the file there is an error, wait the user read
an error window.
    catch
        uiwait(msgbox('The instructions text file ist´n available.'));
        pause(1);
    end
```

instructions.txt

```
------------------------------
Recognition System.

Cesar Juarez Megias.
I.T.T. Imagen y Sonido.
Universidad Carlos III de Madrid.
------------------------------
```

```
This program is based in the SIFT´s code created by Andrea Vedaldi
and Brian Fulkerson in 2007, and posted in the website
http://www.vlfeat.org

You can perform the following tasks with it:

------------------------------
---- TRAINING THE SYSTEM -----
------------------------------
When you go to train the systems, choose a directory where they are
the images.

Sure that the name of the images is the name of the brand logo which
represents the logo image.

------------------------------
-----   RECOGNIZE IMAGE ------
------------------------------
Choose an image to do the logo recognition.

If the resutls are not expected, you can tell the system to add a new
logo.
Be careful with the name you give the logo, because because they can
not change it later.

------------------------------
---  RECOGNIZE DIRECTORY  ----
------------------------------
Choose a directory to do the logo recognition.

Wait the results text file or read the results on the Matlab Command
Window.
```